Interoperable Python ZSI WSDL/SOAP Web Services tutorial

Holger Joukl
LBBW Financial Markets Technologies

21st January 2008

Abstract

This is the follow-up to "Interoperable WSDL/SOAP web services introduction: Python ZSI, Excel XP, gSOAP C/C++ & Applix SS" [7], with a more explicit focus on Python ZSI Web Services.

Now that the "services/SOA" hype & buzzword-storm has calmed down, building SOAP web services servers and clients is widely regarded to have become a commodity. While the complexness of the actual protocols/specs remains, toolkit magic is getting more robust so users seldom need to get down to the nitty-gritty any more. Also, toolkit documentation has broadened and matured. We believe however that a) there can never be enough documentation and b) the intention of this document to be a step-by-step tutorial in the sense of a complete walkthrough still fills a gap.
It features

- as its focus the Python ZSI module (2.0 and 2.1alpha) that is used to build the server side machinery and
- several clients that access the exposed services from
  - Python (ZSI 2.0 + 2.1alpha)
  - C/C++ (gSOAP 2.7.9)

Contents

1 Introduction

This is the follow-up to article [7], mainly prompted by the need to get up-to-date with newer toolkit versions, especially the many changes and improvements in ZSI. While not explicitly stated, ZSI has been the strong focus of this installment from the very beginning, easily noticeable by the fact that all the server code has been Python ZSI-based. Therefore, this document focuses on ZSI and reduces the number of discussed Web Service client implementations in comparison to [7].

The toolkit versions discussed here are:

- ZSI 2.0 + 2.1alpha1 (Python 2.4.4)
- gSOAP 2.7.9 (gcc 2.95.2, gcc 3.4.4)

These are being put to use for several example Web Services.

For legacy reference, the steps to instrument VistaSource Applixware 4.43 with Web Services capabilities (which is probably of minor interest to the average ZSI/gSOAP user) have been updated for the few gSOAP handling changes and still remain available in appendix A.

Appendix B features a (arguably nifty) way to make ZSI tracing more verbose.

1.1 Goals & Concepts

This is a practice-report-gone-tutorial. The content and code examples presented here have been developed with the primary goal to "make (fairly) simple examples work (at all)", in a learning-by-doing manner. Thus there is lots of room for enhancements, e.g. getting rid of hardcoded path names etc.

All code examples are complete, executable and delivered with all the necessary command-line options to create and/or compile and/or run them.

Conceptually, we use a WSDL-centric approach, sometimes referred to as top-down development: The starting point for all example service and client implementations will be the WSDL description. Note that this might differ from certain toolkits that start out with the service implementation in the host language and generate the WSDL for you to expose the implemented service. We regard the latter to have a tendency to not promote interoperability and to tie in implementation language details, which is certainly not what we want.[1]

Striving for interoperability, only the WS-I-compliant[2] rpc/literal and document/literal WSDL styles are presented here.

Throughout this document, certain host names or ports ("8080") might used in the examples - you will have to substitute those with the appropriate setup for your site, of course. Naturally, this can affect URLs defined in the example WSDLs and used to retrieve these WSDLs.

Whenever you see sample client or server output throughout this document this is real output copied from live components but might have been reformatted to fit into the document somewhat nicer.

[1]In the first place, this came up partly due to the fact that the chosen server implementation (Python ZSI) offered no such tool and partly as a gut feeling. Since then, this opinion has grown stronger and has also been backed up by several practitioners´ readings at a conference (Stuttgarter Softwaretechnik Forum 2005, Stuttgart-Vaihingen, Germany) as well as experience.

[2]The **W**eb **S**ervices-**I**nteroperability Organization (www.ws-i.org) provides so-called "profile" guidelines on using Web Services specifications in an interoperable manner.

Acknowledgments: Thank you to the ZSI developers for supporting a great product!

The DateService WSDL (sect. 3.1) and worker code (sect. 3.2) are courtesy of Rainer Kluger (LBBW Financial Markets Technologies).

1.2 Prerequisites

We assume the reader is familiar with Python and/or C/C++ to a certain degree. The web service server components are implemented in Python, so to build a working server with the code samples some Python knowledge is needed, but any of the client side technologies can be skipped if not of particular interest.

While some basic concepts regarding WSDL, SOAP, HTTP servers are presented here implicitly, this document is not a tutorial on these. If you want to know more there´s plenty of stuff on the web.

1.3 A glance at the toolkits

1.3.1 ZSI

The Python ZSI package [1] is the actively maintained one of two "pywebsvcs"[3] packages[4] implementing web services for Python, namely SOAP 1.1 messaging and WSDL capabilities. It is powerful and easy to get started with. With the arrival of ZSI 2.0 not only have the ZSI developers honed the features and ease-of-use of their fine product but also the documentation has enhanced quite a bit (see especially the ZSI user guide[2]). As always, there´s still room for improvement, and we hope to provide some added value by presenting simple, concise examples here.

At the time of writing, ZSI is in flux again with regard to a planned move to WSGI[5] support, which 2.1alpha1 already gives a notion of, and certain code generation/naming conventions. For the time being, we rely on "classic" ZSI usage here, presenting both 2.0 and 2.1alpha code versions in our examples. All code denoted as ZSI 2.1 refers to 2.1alpha1-compatible code, as of release 2.1alpha1 (2007-11-01). As a side note, ZSI 2.1 gains a massive performance boost from switching to minidom as default parser, removing the dependency on PyXML.

1.3.2 gSOAP

gSOAP is an impressive open source web services development toolkit for C/C++. It seems to be very mature and complete and has an extensive record of being used in real-world applications by major companies. One of its key-features is the support for XML-to-C/C++ mapping for native C and C++ data types. It claims to be the fastest SOAP 1.1/1.2 compliant C/C++ implementation out there and comes with good documentation.

2 Simple datatypes: The rpc/literal SquareService

This first example will implement an overly simple service that exposes a function which takes a `double` argument and returns the square of it (x^2) as a `double`. I.e. this examples uses simple scalar datatypes, one single argument and one single return value.

2.1 The SquareService WSDL

This is the WSDL file that determines the contract for the SquareService, called `SquareService.wsdl`:

[3]http://pywebsvcs.sourceforge.net/
SourceForge project page:
http://sourceforge.net/projects/pywebsvcs
[4]The other being SOAPpy.
[5]Python Web Server Gateway Interface v1.0

```
<?xml version="1.0"?>
<definitions name="SquareService"
 targetNamespace="http://services.zsiserver.net:8080/SquareService"
 xmlns:tns="http://services.zsiserver.net:8080/SquareService"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

    <message name="getSquareRequest">
        <part name="x" type="xsd:double"/>
    </message>
    <message name="getSquareResponse">
        <part name="return" type="xsd:double"/>
    </message>

    <portType name="SquarePortType">
        <operation name="getSquare">
            <documentation> the square method </documentation>
            <input message="tns:getSquareRequest"/>
            <output message="tns:getSquareResponse"/>
        </operation>
    </portType>

    <binding name="SquareBinding" type="tns:SquarePortType">
        <soap:binding style="rpc" trans-
port="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="getSquare">
            <soap:operation soa-
pAction="http://services.zsiserver.net:8080/SquareService/getSquare"/>
            <input>
                <soap:body use="literal" names-
pace="http://services.zsiserver.net:8080/SquareService"/>
            </input>
            <output>
                <soap:body use="literal" names-
pace="http://services.zsiserver.net:8080/SquareService"/>
            </output>
        </operation>
    </binding>

    <service name="SquareService">
        <documenta-
tion>Returns x^2 (x**2, square(x)) for a given float x</documentation>
        <port name="SquarePort" binding="tns:SquareBinding">
            <soap:address loca-
tion="http://services.zsiserver.net:8080/SquareService"/>
        </port>
    </service>

</definitions>
```

Comments:

- The `style` “rpc” and the `use` “literal” are used, to be WS-I-compliant. WS-I only supports rpc/literal and document/literal.
- A custom port 8080 is used instead of the usual HTTP port 80.
- The server name is a “symbolic” name, not a real known name in our site. This can cause the need to give explicit URLs from time to time.

2.2 A Python ZSI server for the SquareService

2.2.1 Generating stubs from WSDL

ZSI 2.1 ZSI 2.1 comes with a python script `wsdl2py` to generate code from a WSDL file. It creates

- python bindings for the service and data structures and
- a server skeleton for service dispatch where the actual service worker code will be hooked into.

If you have installed ZSI on top of your python installation you can invoke the scripts like this (change your installation base path according to your setup):[6]

```
/apps/pydev/hjoukl/bin/wsdl2py SquareService.wsdl
```

This will generate the files

- `SquareService_client.py`,
- `SquareService_server.py` and
- `SquareService_types.py`.

ZSI 2.0 ZSI 2.0 uses the two scripts `wsdl2py` and `wsdl2dispatch` for code generation and uses different names for the generated files:

```
$ /apps/pydev/bin/wsdl2py --file SquareService.wsdl
$ /apps/pydev/bin/wsdl2dispatch --file SquareService.wsdl
$ $ ls
SquareService.wsdl@        SquareService_services_server.py
SquareService_services.py  SquareService_services_types.py
```

What do we have now? We have bindings to work with the services in python and a skeleton for dispatching to the actual worker methods. What we still need is

- the main program that runs a (HTTP-) server with a request handler for the services and
- the hooks to invoke the worker methods.

Luckily, ZSI includes the ZSI.ServiceContainer module which implements all this machinery for us.

2.2.2 Writing the SquareService web server

This is our main program `mySquareServer.py`. It basically imports the necessary ZSI stuff, adds minimal command line configurability for logging purposes and lets us choose a server HTTP port:

ZSI 2.1

```
#!/apps/pydev/hjoukl/bin/python2.4

# *** ZSI 2.1 ***

from optparse import OptionParser

# Import the ZSI stuff you'd need no matter what
from ZSI.wstools import logging
```

[6]The installation base path for all ZSI 2.1 examples here is /apps/pydev/hjoukl.

```
from ZSI.ServiceContainer import AsServer

# Import the generated Server Object
from SquareService_server import SquareService


# Create a Server implementation by inheritance
class MySquareService(SquareService):

    # Make WSDL available for HTTP GET
    _wsdl = "".join(open("SquareService.wsdl").readlines())

    def soap_getSquare(self, ps, **kw):
        # Call the generated base class method to get appropriate
        # input/output data structures
        request, response = SquareService.soap_getSquare(self, ps, **kw)
        response._return = self.getSquare(request._x)
        return request, response

    def getSquare(self, x):
        return x**2

op = OptionParser(usage="%prog [options]")
op.add_option("-l", "--loglevel", help="loglevel (DEBUG, WARN)",
              metavar="LOGLEVEL")
op.add_option("-p", "--port", help="HTTP port",
              metavar="PORT", default=8080, type="int")
options, args = op.parse_args()

# set the loglevel according to cmd line arg
if options.loglevel:
    loglevel = eval(options.loglevel, logging.__dict__)
    logger = logging.getLogger("")
    logger.setLevel(loglevel)

# Run the server with a given list services
AsServer(port=options.port, services=[MySquareService(),])
```

ZSI 2.0

```
#!/apps/pydev/bin/python2.4
# *** ZSI 2.0 ***
from optparse import OptionParser

# Import the ZSI stuff you'd need no matter what
from ZSI.wstools import logging
from ZSI.ServiceContainer import AsServer

# Import the generated Server Object
from SquareService_services_server import SquareService


# Create a Server implementation by inheritance
class MySquareService(SquareService):

    # Make WSDL available for HTTP GET
    _wsdl = "".join(open("SquareService.wsdl").readlines())

    def soap_getSquare(self, ps, **kw):
        # Call the generated base class method to get appropriate
        # input/output data structures
```

```
            response = SquareService.soap_getSquare(self, ps, **kw)
            request = self.request
            response._return = self.getSquare(request._x)
            return response

        def getSquare(self, x):
            return x**2

    op = OptionParser(usage="%prog [options]")
    op.add_option("-l", "--loglevel", help="loglevel (DEBUG, WARN)",
                  metavar="LOGLEVEL")
    op.add_option("-p", "--port", help="HTTP port",
                  metavar="PORT", default=8080, type="int")
    options, args = op.parse_args()

    # set the loglevel according to cmd line arg
    if options.loglevel:
        loglevel = eval(options.loglevel, logging.__dict__)
        logger = logging.getLogger("")
        logger.setLevel(loglevel)

    # Run the server with a given list services
    AsServer(port=options.port, services=[MySquareService(),])
```

Apart from the command line parsing stuff which is only usability sugar, ZSI provides us with everything we need. All we have to do is

1. Create a class `MySquareService` that inherits from the generated `SquareService` class, to hook in the worker code. This is done with the overridden `soap_getSquare()` method, which

    - calls its base class method to retrieve the (input and) output data structure (i.e. request & response) and
    - invokes the worker code that performs the actual calculation, and populates the response data structure. Note that the names in these data structure differ from the names in the WSDL in that they use leading underscores. This is done with good reason (e.g., you can not use the name "return" for the result message part `<part name="return" type="xsd:double"/>` as a valid Python attribute name) due to XML allowing a far wider range of element names than the Python language. You can find details on ZSI naming concepts ("aname") in the ZSI User Guide [2].

    Here´s the only difference between the 2.0 & 2.1 implementations, as 2.1 slightly changes the soap_<...> methods signature and return value.

    We also add a _wsdl attribute that contains the WSDL, to be served when some client issues a HTTP GET (on `<service URL>?wsdl`), which is actually common behaviour to get the service description (apart from UDDI).[7] ZSI 2.0 wouldn´t need this additional step as its code generation mechanisms put all the WSDL source into the generated files anyway, but 2.1 does not (and we consider 2.1 behaviour cleaner).

2. Put an instance of our class into the `services` argument of the `AsServer()` function.

That´s it. We can now run a full-fledged WS-Server that hosts our SquareService.

Comments:

- The dispatch to the appropriate service operation is handled in the `soapAction` dictionary in the generated code. This dictionary maps the action that is requested to the method that gets invoked. The ZSI standard request handler takes the HTTP header field soapAction and propagates its value to this dispatch mechanism. Thus, everything works out-of-the-box if you use the "soapAction" operation-attribute in your WSDL file (and if your service client actually provides this header field with its request). We´ll see another way to drive the ZSI request handler in section 4.2.

[7]The service invocation itself uses HTTP POST. You might want to take a look at the ZSI ServiceContainer module and its `SOAPRequestHandler` class for more inside information.

- Command-line options allow us to make use of ZSI´s logging capabilities, to print out more information on ZSI server workings (unfortunately, not the incoming XML request. See section B.2 for help on this).

2.3 A Python ZSI client for the SquareService

2.3.1 Using generated client code

We implement a client that calls getSquare from the SquareService in `mySquareClient.py` as follows:

ZSI 2.1

```
#!/apps/pydev/hjoukl/bin/python2.4
# *** ZSI 2.1 ***
import sys
from optparse import OptionParser

# Import the generated client code
from SquareService_client import *


op = OptionParser(usage="%prog [options]")
op.add_option("-u", "--url", help="service URL", metavar="URL")

options, args = op.parse_args()

service = SquareServiceLocator().getSquarePort(url=options.url, trace-
file=sys.stdout)

print '\nAccessing service SquareService, method getSquare...'
while 1:
    x = float(raw_input("Enter x: "))

    request = getSquareRequest(x=x)

    response = service.getSquare(request)

    print "x**2 =", response._return
```

ZSI 2.0

```
#!/apps/pydev/bin/python2.4

# *** ZSI 2.0 ***

import sys
from optparse import OptionParser

# Import the generated client code
from SquareService_services import *


op = OptionParser(usage="%prog [options]")
op.add_option("-u", "--url", help="service URL", metavar="URL")

options, args = op.parse_args()

service = SquareServiceLocator().getSquarePortType(url=options.url, trace-
file=sys.stdout)
```

```
print '\nAccessing service SquareService, method getSquare...'
while 1:
    x = float(raw_input("Enter x: "))

    request = getSquareRequest()
    request._x = x

    response = service.getSquare(request)

    print "x**2 =", response._return
```

This is pretty straightforward. Some of the code handles command line stuff which has nothing to do with web services in the first place. The things we have to do is to

1. Import the generated code we need, from either the `SquareService_client` (ZSI 2.1) / `SquareService_services` (ZSI 2.0) module,
2. Create a `SquareServiceLocator`,
3. Invoke its `getSquarePort()` (ZSI 2.1) / `getSquarePortType()` (ZSI 2.0)method to retrive the square service binding,
4. populate the request data structure
   - using keyword-arguments (ZSI 2.1)
   - by explicitly setting request instance attributes (ZSI 2.0)
5. call the service binding´s `getSquare()` method with the request data as input parameter and
6. read the result data structure.

Again, the naming concepts of ZSI as noted in section 2.2.2 apply.

This is a sample session of our new client:

```
$ ./mySquareClient.py -u "http://adevp02:8080/SquareService"

Accessing service SquareService, method getSquare...
Enter x: 3
_________________________________ Wed Jan  2 10:16:23 2008 REQUEST:
<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header><SOAP-ENV:Body
xmlns:ns1="http://services.zsiserver.net/SquareService">
<ns1:getSquare>
<x>3.000000</x>
</ns1:getSquare>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
_________________________________ Wed Jan  2 10:16:23 2008 RESPONSE:
200
OK
-------
Server: ZSI/1.1 BaseHTTP/0.3 Python/2.4.4
Date: Wed, 02 Jan 2008 09:16:23 GMT
Content-type: text/xml; charset="utf-8"
Content-Length: 497
```

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body
xmlns:ns1="http://services.zsiserver.net/SquareService">
<ns1:getSquareResponse>
<return>9.000000</return>
</ns1:getSquareResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
x**2 = 9.0
Enter x:
```

Comments:

- We use an explicit URL to access the service, as our WSDL contains only the "symbolic" server name not known by our site´s naming services. In a real-world application, you'd not even need to do so, as the generated client code knows the service URL as given in the WSDL.
- The client does not issue any HTTP GET for service access, because all the necessary data structures have been pre-generated from the WSDL service description.

2.3.2 Using ServiceProxy

ZSI also offers an option to implement a client without using the pre-generated stub code: Everything can be accessed through the `ServiceProxy` object that gets all the necessary information from the WSDL, on the fly. This is an example of a ServiceProxy-client (the very same code works with both ZSI 2.1 & 2.0):

```
#!/apps/pydev/bin/python2.4

import sys
from optparse import OptionParser

from ZSI.ServiceProxy import ServiceProxy

op = OptionParser(usage="%prog [options]")
op.add_option("-u", "--url", help="service URL (mandatory)", metavar="URL")

options, args = op.parse_args()
if not options.url:
    op.print_help()
    sys.exit(1)
else:
    # "Canonicalize" WSDL URL
    options.url = options.url.replace("?WSDL", "?wsdl")
    if not options.url.endswith("?wsdl"):
        options.url += "?wsdl"

    service = ServiceProxy(wsdl=options.url, tracefile=sys.stdout)

    print '\nAccessing service SquareService, method getSquare...'
    while 1:
        x = float(raw_input("Enter x: "))
        resultDict = service.getSquare(x=x)
        print "x**2 =", resultDict['return']
```

Basically, all you need to do is to create a `ServiceProxy` instance with the information where to get the WSDL. You can then simply access its getSquare method; only make sure you give its argument as a keyword argument.

This is the output of an example ServiceProxy-client run:

```
$ ./mySquareSPclient.py -u "http://adevp02:8080/SquareService?wsdl"

Accessing service SquareService, method getSquare...
Enter x: 4
_________________________________ Wed Jan  2 10:30:17 2008 REQUEST:
<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:ns1="http://services.zsiserver.net/SquareService">
<ns1:getSquare>
<x xsi:type="xsd:double">4.000000</x>
</ns1:getSquare>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
_________________________________ Wed Jan  2 10:30:18 2008 RESPONSE:
200
OK
-------
Server: ZSI/1.1 BaseHTTP/0.3 Python/2.4.4
Date: Wed, 02 Jan 2008 09:30:17 GMT
Content-type: text/xml; charset="utf-8"
Content-Length: 498

<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header><SOAP-
ENV:Body xmlns:ns1="http://services.zsiserver.net/SquareService">
<ns1:getSquareResponse>
<return>16.000000</return>
</ns1:getSquareResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
x**2 = 16.0
Enter x:
```

Comments:

- When running the ServiceProxy-notion of a ZSI WS-client, take a good look at your ZSI server. You will then see that the ServiceProxy client issues an HTTP GET first, to retrieve the WSDL information and create the necessary code and data structures.
- To tell the whole truth, ServiceProxy basically does much the same code generation like `wsdl2py` under the hood, and creates a cache dir `~/.zsi_service_proxy_dir` (ZSI 2.1) resp. `./.service_proxy_dir` (ZSI 2.0) for the generated files. This cache dir is then appended to `sys.path` for module import.[8]

[8]The attentive reader might observe that this may have an implication on ServiceProxy-client behaviour depending on where you run it. As the local directory is the 1st `sys.path` entry per default, existing pre-generated (from `wsdl2py`) code might beat the cache-dir modules in import order - depending on module naming.

2.4 A gSOAP C++ client for the SquareService

2.4.1 Generation from WSDL

With our SquareService server running, we can generate the client stubs:

1. First, gSOAP needs to create a header file for the service using the `wsdl2h` generator:

   ```
   /apps/pydev/bin/wsdl2h -o squareService.h
   http://adevp02:8080/SquareService?wsdl
   ```

   Just specify the name for the header file and the URL where the WSDL can be received with a HTTP GET request. The `squareService.h` header will be created:

   ```
   /apps/pydev/bin/wsdl2h -
   o squareService.h http://adevp02:8080/SquareService?wsdl

   **  The gSOAP WSDL parser for C and C++ 1.2.9l
   **  Copyright (C) 2000-2007 Robert van Engelen, Genivia Inc.
   **  All Rights Reserved. This product is provided "as is", without any war-
   ranty.
   **  The gSOAP WSDL parser is released under one of the following two li-
   censes:
   **  GPL or the commercial license by Genivia Inc. Use option -
   l for more info.

   Saving squareService.h

   Cannot open file 'typemap.dat'
   Problem reading type map file typemap.dat.
   Using internal type definitions for C++ instead.

   Connecting to 'http://adevp02:8080/SquareService?wsdl' to re-
   trieve WSDL/XSD... connected, receiving...
   Warning: part 'return' uses literal style and should refer to an ele-
   ment rather than a type
   Warning: part 'x' uses literal style and should refer to an ele-
   ment rather than a type
   Warning: part 'x' uses literal style and should refer to an ele-
   ment rather than a type
   Warning: part 'x' uses literal style and should refer to an ele-
   ment rather than a type

   To complete the process, compile with:
   soapcpp2 squareService.h
   ```

   Note that gSOAP tells us about what it thinks might cause problems in our WSDL; we ignore that for this example.

2. Next we let gSOAP create the stub code for us, using the newly-created header:

   ```
   $ /apps/pydev/bin/soapcpp2 -I /data/pydev/DOWNLOADS/WebServices/C++/gsoap-
   2.7/soapcpp2/import  squareService.h

   **  The gSOAP Stub and Skeleton Compiler for C and C++ 2.7.9l
   **  Copyright (C) 2000-2007, Robert van Engelen, Genivia Inc.
   **  All Rights Reserved. This product is provided "as is", without any war-
   ranty.
   **  The gSOAP compiler is released under one of the following three li-
   censes:
   **  GPL, the gSOAP public license, or the commercial license by Genivia Inc.
   ```

```
Saving soapStub.h
Saving soapH.h
Saving soapC.cpp
Saving soapClient.cpp
Saving soapClientLib.cpp
Saving soapServer.cpp
Saving soapServerLib.cpp
Using ns1 service name: SquareBinding
Using ns1 service style: document
Using ns1 service encoding: literal
Using ns1 service location: http://adevp02:8080/SquareService
Using ns1 schema namespace: http://services.zsiserver.net/SquareService
Saving soapSquareBindingProxy.h client proxy
Saving soapSquareBindingObject.h server object
Saving SquareBinding.getSquare.req.xml sample SOAP/XML request
Saving SquareBinding.getSquare.res.xml sample SOAP/XML response
Saving SquareBinding.nsmap namespace mapping table

Compilation successful
```

We must explicitly give the `gSOAP/soapcpp2` directory as include directory. This is not being installed with the gSOAP installation (in the "make install" step) but resides in the path where you extracted the gSOAP tarball. It contains some special header files gSOAP does not install on your system.

You might have noticed that `soapcpp2` says it is using service style "document" as opposed to what´s defined in the WSDL ("rpc"); this seems to be a cosmetic issue only and does not affect the usability of the generated code.

The above command produces the following client stubs (server skeleton code also by the way, but we will not use it here):

```
$ ls -l
total 105
-rw-r--r--
1 lb54320 intern   444 Jan  2 10:48 SquareBinding.getSquare.req.xml
-rw-r--r--
1 lb54320 intern   470 Jan  2 10:48 SquareBinding.getSquare.res.xml
-rw-r--r--  1 lb54320 intern   556 Jan  2 10:48 SquareBinding.nsmap
-rw-r--r--  1 lb54320 intern 57364 Jan  2 10:48 soapC.cpp
-rw-r--r--  1 lb54320 intern  2326 Jan  2 10:48 soapClient.cpp
-rw-r--r--  1 lb54320 intern   464 Jan  2 10:48 soapClientLib.cpp
-rw-r--r--  1 lb54320 intern 15694 Jan  2 10:48 soapH.h
-rw-r--r--  1 lb54320 intern  3152 Jan  2 10:48 soapServer.cpp
-rw-r--r--  1 lb54320 intern   464 Jan  2 10:48 soapServerLib.cpp
-rw-r--r--  1 lb54320 intern  2470 Jan  2 10:48 soapSquareBindingObject.h
-rw-r--r--  1 lb54320 intern  1826 Jan  2 10:48 soapSquareBindingProxy.h
-rw-r--r--  1 lb54320 intern  6915 Jan  2 10:48 soapStub.h
-rw-r--r--  1 lb54320 intern  6662 Jan  2 10:38 squareService.h
```

What´s left now is to implement the client program and make use of the generated code.

2.4.2 Client implementation

This is a sample client to access the SquareService:

```
$ cat myCSquareClient.cpp
#include "soapH.h"
#include "SquareBinding.nsmap"

#include <iostream>
```

```
int main(void)
{
    struct soap soap;
    double x = 0;
    struct ns1__getSquareResponse response;

    soap_init(&soap);
    while (1) {
        std::cout << "Enter x value: ";
        std::cin >> x;
        if (soap_call_ns1__getSquare(&soap, NULL, NULL, x, re-
sponse) == SOAP_OK) {
            std::cout << "Result: " << response.return_ << std::endl;
        } else {
            soap_print_fault(&soap, stderr);
        }
    }
    soap_destroy(&soap);
    soap_end(&soap);
    soap_done(&soap);
    return 0;
}
```

Comments:

- Note gSOAP´s renaming of the WSDL result message part `<part name="return" type="xsd:double"/>` to `return_` in the response data structure `struct ns1__getSquareResponse` to cater for conflicts with reserved C/C++ words, similar to the ZSI "aname" concept
- There are also other ways to invoke this service with the gSOAP mechanisms, namely the `Square-Binding` class defined in `soapSquareBindingProxy.h`, which would take care of the initialization & destruction activities needed in the above client code. We will use this (better) approach in the next examples.

2.4.3 gSOAP client compilation

gcc 2.95.2

```
$ g++ -o myCSquareClient -R/apps/prod/lib -I/apps/pydev/include
-L/apps/pydev/lib soapC.cpp soapClient.cpp myCSquareClient.cpp -lgsoap++ -
lsocket
```

gcc 3.4.4 Note: Compiling with gcc 3.4.4, the nsl library had to be added to the linked libraries:

```
/apps/local/gcc/3.4.4/bin/g++ -o myCSquareClient -R /apps/prod/gcc/3.4.4/lib -
I/apps/pydev/gcc/3.4.4/include -
L/apps/pydev/gcc/3.4.4/lib soapC.cpp soapClient.cpp myCSquareClient.cpp -
lgsoap++ -lsocket -lnsl
```

You can then run this simple C++ Web Service client:

```
$ ./myCSquareClient
Enter x value: 3
Result: 9
Enter x value: ^C
```

3 Structured datatypes: The rpc/literal DateService

Let´s move on to a more elaborate service, elaborate in the sense of using structured datatypes now (not that the service example itself was particularly ingenious). Anyway, we will now implement the DateService service that exposes two methods:

- <date structure> getCurrentDate(<string>) takes a string argument and returns the current date as a datetime structure
- <date structure> getDate(<int>, <date structure>) takes an integer offset and a date structure as arguments and returns the given date plus the offset (in days) as a date structure

3.1 The DateService WSDL

The DateService is described in `DateService.wsdl`:

```
<?xml version="1.0"?>
<definitions name="DateService"
  targetNamespace="http://services.zsiserver.net/DateService.wsdl"
  xmlns:tns="http://services.zsiserver.net:8080/DateService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:myType="DateType_NS"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace="DateType_NS" >
      <complexType name="Date">
            <sequence>
              <element name="year" nillable="true" type="xsd:integer"/>
              <element name="month" nillable="true" type="xsd:integer"/>
              <element name="day" nillable="true" type="xsd:integer"/>
              <element name="hour" nillable="true" type="xsd:integer"/>
              <element name="minute" nillable="true" type="xsd:integer"/>
              <element name="second" nillable="true" type="xsd:integer"/>
              <element name="weekday" nillable="true" type="xsd:integer"/>
              <element name="dayOfYear" nillable="true" type="xsd:integer"/>
              <element name="dst" nillable="true" type="xsd:integer"/>
            </sequence>
      </complexType>
    </schema>
  </types>
  <message name="getCurrentDateRequest">
    <part name="input" type="xsd:string"/>
  </message>
  <message name="getCurrentDateResponse">
    <part name="today" type="myType:Date"/>
  </message>
  <message name="getDateRequest">
    <part name="offset" type="xsd:integer"/>
    <part name="someday" type="myType:Date"/>
  </message>
  <message name="getDateResponse">
    <part name="day" type="myType:Date"/>
  </message>


  <portType name="DateService_PortType">
```

```
    <operation name="getCurrentDate">
      <input message="tns:getCurrentDateRequest"/>
      <output message="tns:getCurrentDateResponse"/>
    </operation>
    <operation name="getDate">
      <input message="tns:getDateRequest"/>
      <output message="tns:getDateResponse"/>
    </operation>
  </portType>


  <binding name="DateService_Binding" type="tns:DateService_PortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getCurrentDate">
      <soap:operation soapAction="urn:DateService.wsdl#getCurrentDate"/>
      <input>
        <soap:body use="literal" namespace="urn:DateService.wsdl"/>
      </input>
      <output>
        <soap:body use="literal" namespace="urn:DateService.wsdl"/>
      </output>
    </operation>
    <operation name="getDate">
      <soap:operation soapAction="urn:DateService.wsdl#getDate"/>
      <input>
        <soap:body parts="offset someday" use="literal" names-
pace="urn:DateService.wsdl"/>
      </input>
      <output>
        <soap:body use="literal" namespace="urn:DateService.wsdl"/>
      </output>
    </operation>
  </binding>


  <service name="simple Date Service">
    <documentation>Date Web Service</documentation>
    <port name="DateService_Port" binding="tns:DateService_Binding">
      <soap:address location="http://services.zsiserver.net/DateService"/>
    </port>
  </service>

</definitions>
```

Comments:

- Again, rpc/literal has been chosen.
- A ComplexType "Date" is defined in the <types> section. This type is being used as a return type (getCurrentDate, getDate) and as a method argument type (getDate).

3.2 A Python ZSI DateService server

The tasks at hand are the same as for the SquareService example.

3.2.1 Code generation from WSDL

ZSI 2.1

```
$ /apps/pydev/hjoukl/bin/wsdl2py DateService.wsdl
$ ls -l
total 15
-rw-r--r--  1 lb54320 intern 3180 Jan  2 14:12 DateService.wsdl
-rw-r--r--  1 lb54320 intern 3935 Jan  2 14:12 DateService_client.py
-rw-r--r--  1 lb54320 intern 3209 Jan  2 14:12 DateService_server.py
-rw-r--r--  1 lb54320 intern 2896 Jan  2 14:12 DateService_types.py
```

ZSI 2.0

```
$ /apps/pydev/bin/wsdl2py -f DateService.wsdl
$ /apps/pydev/bin/wsdl2dispatch -f DateService.wsdl
$ ls -l
total 34
lrwxrwxrwx  1 lb54320 intern   28 Jan  3 17:45 DateService.wsdl -
> ../2.1alpha/DateService.wsdl
-rw-r--r--  1 lb54320 intern 3525 Jan  7 09:56 DateService_services.py
-rw-r--r--  1 lb54320 intern 4428 Jan  7 09:56 DateService_services_server.py
-rw-r--r--  1 lb54320 intern 2818 Jan  7 09:56 DateService_services_types.py
```

3.2.2 The DateService server-side implementation

The server implementation needs exactly the same steps as seen in section 2.2.2, with the difference of putting a DateService instance into the ServiceContainer now. We create a server `myDateServer.py` like this:

ZSI 2.1

```
#!/apps/pydev/hjoukl/bin/python2.4
import time
from optparse import OptionParser
# Import the ZSI stuff you'd need no matter what
from ZSI.wstools import logging
from ZSI.ServiceContainer import AsServer
# Import the generated Server Object
from DateService_server import *
# Create a Server implementation by inheritance
class MyDateService(simple_Date_Service):
   # Make WSDL available for HTTP GET
    _wsdl = "".join(open("DateService.wsdl").readlines())
   def soap_getCurrentDate(self, ps, **kw):
        # Call the generated base class method to get appropriate
        # input/output data structures
        request, response = sim-
ple_Date_Service.soap_getCurrentDate(self, ps, **kw)
        print "Client says:", request._input
        dt = time.localtime(time.time())
       # Use a structurally equivalent (to the WSDL complex type structure)
        # python class object
        class today:
            _year = dt[0]
            _month = dt[1]
            _day = dt[2]
            _hour = dt[3]
            _minute = dt[4]
            _second = dt[5]
            _weekday = dt[6]
            _dayOfYear = dt[7]
            _dst = dt[8]
        response._today = today
```

```
            return request, response
        def soap_getDate(self, ps, **kw):
            # Call the generated base class method to get appropriate
            # input/output data structures
            request, response = simple_Date_Service.soap_getDate(self, ps, **kw)
            date = request._someday
            offset = request._offset
            if not offset:
                offset = 0
            if not date:
                raise RuntimeError("missing input data")

            # actual worker code, add give offset to given input date
            sec = 3600 * 24  # seconds/hour * 24h
            providedDate_tuple = (date._year, date._month, date._day,
                                  date._hour, date._minute, date._second,
                                  date._weekday, date._dayOfYear, date._dst)
            providedDate_sec = time.mktime(providedDate_tuple)
            offset_sec = sec * offset
            newDate_sec = providedDate_sec + offset_sec
            newDate_tuple = time.localtime(newDate_sec)

            response = getDateResponse()
            # Use a structurally equivalent (to the WSDL complex type structure)
            # python class object
            class day:
                _year = newDate_tuple[0]
                _month = newDate_tuple[1]
                _day = newDate_tuple[2]
                _hour = newDate_tuple[3]
                _minute = newDate_tuple[4]
                _second = newDate_tuple[5]
                _weekday = newDate_tuple[6]
                _dayOfYear = newDate_tuple[7]
                _dst = newDate_tuple[8]
            response._day = day

            return request, response


    op = OptionParser(usage="%prog [options]")
    op.add_option("-l", "--loglevel", help="loglevel (DEBUG, WARN)",
                  metavar="LOGLEVEL")
    op.add_option("-p", "--port", help="HTTP port",
                  metavar="PORT", default=8080, type="int")
    options, args = op.parse_args()

    # set the loglevel according to cmd line arg
    if options.loglevel:
        loglevel = eval(options.loglevel, logging.__dict__)
        logger = logging.getLogger("")
        logger.setLevel(loglevel)

    # Run the server with a given list services
    AsServer(port=options.port, services=[MyDateService(),])
```

As in the previous example, the actual implementation has been hooked into the server skeleton, by inheriting from the generated service class.

If you take a closer look at the two method implementations, you will notice that the data structures used for the returned date are just (nested) python classes. ZSI handles the serialization of that class into the actual SOAP message for us.[9]In

Comments:

- This time, we did not strictly separate the worker code from the `soap_<...>` methods.
- The handling of complex types can be made more convenient by using the –complexType option of wsdl2py. This is the recommended way to go and we´ll see how it works in section 4, but it requires metaclass magic and thus new-style classes to work, so you will require Python >= 2.2.

ZSI 2.0 Remembering the SquareService example, the ZSI 2.1 and 2.0 server code differed only in the return value and method signature of the `soap_<...>()`-method implementation-wise. We can exploit this to reuse the code written for ZSI 2.1, by help of an adapter module that provides some facilities to auto-wrap these methods. Using this adapter module, we can now implement a working ZSI 2.0 server like this:

```
import time
from optparse import OptionParser
# Import the ZSI stuff you'd need no matter what
from ZSI.wstools import logging
from ZSI.ServiceContainer import AsServer
# Import the generated Server Object
from ZSI.version import Version as zsiversion
print "*** ZSI version %s ***" % (zsiversion,)
# Import the generated Server Object
try:
    # 2.1alpha
    from DateService_server import simple_Date_Service
    # Dummy implementation
    def adapt(method):
        return method
except ImportError, e:
    # 2.0final
    from DateService_services_server import simple_Date_Service

    from service_adaptor import ServiceAdaptor21
    adapt = ServiceAdaptor21.adapt

    # Wrap generated 2.0 class to 2.1 soap_ method behaviour
    simple_Date_Service = ServiceAdaptor21(simple_Date_Service)
# Create a Server implementation by inheritance
# 2.1alpha implementation
class MyDateService(simple_Date_Service):

    # Make WSDL available for HTTP GET
    _wsdl = "".join(open("DateService.wsdl").readlines())

    @adapt
    def soap_getCurrentDate(self, ps, **kw):
        # Call the generated base class method to get appropriate
        # input/output data structures
        request, response = sim-
ple_Date_Service.soap_getCurrentDate(self, ps, **kw)
        print "Client says:", request._input
        dt = time.localtime(time.time())
        # Use a structurally equivalent (to the WSDL complex type structure)
        # python class object
        class today:
            _year = dt[0]
```

[9]The python object must be "structurally equivalent" to the XML datatype that is defined in the WSDL and that constitutes ZSI´s typecodes.

```
            _month = dt[1]
            _day = dt[2]
            _hour = dt[3]
            _minute = dt[4]
            _second = dt[5]
            _weekday = dt[6]
            _dayOfYear = dt[7]
            _dst = dt[8]
        response._today = today
        return request, response
    @adapt
    def soap_getDate(self, ps, **kw):
        # Call the generated base class method to get appropriate
        # input/output data structures
        request, response = simple_Date_Service.soap_getDate(self, ps, **kw)
        date = request._someday
        offset = request._offset
        if not offset:
            offset = 0
        if not date:
            raise RuntimeError("missing input data")

        # actual worker code, add give offset to given input date
        sec = 3600 * 24  # seconds/hour * 24h
        providedDate_tuple = (date._year, date._month, date._day,
                              date._hour, date._minute, date._second,
                              date._weekday, date._dayOfYear, date._dst)
        providedDate_sec = time.mktime(providedDate_tuple)
        offset_sec = sec * offset
        newDate_sec = providedDate_sec + offset_sec
        newDate_tuple = time.localtime(newDate_sec)

        # Use a structurally equivalent (to the WSDL complex type structure)
        # python class object
        class day:
            _year = newDate_tuple[0]
            _month = newDate_tuple[1]
            _day = newDate_tuple[2]
            _hour = newDate_tuple[3]
            _minute = newDate_tuple[4]
            _second = newDate_tuple[5]
            _weekday = newDate_tuple[6]
            _dayOfYear = newDate_tuple[7]
            _dst = newDate_tuple[8]
        response._day = day

        return request, response


op = OptionParser(usage="%prog [options]")
op.add_option("-l", "--loglevel", help="loglevel (DEBUG, WARN)",
              metavar="LOGLEVEL")
op.add_option("-p", "--port", help="HTTP port",
              metavar="PORT", default=8080, type="int")
options, args = op.parse_args()

# set the loglevel according to cmd line arg
if options.loglevel:
    loglevel = eval(options.loglevel, logging.__dict__)
    logger = logging.getLogger("")
    logger.setLevel(loglevel)
# Run the server with a given list services
AsServer(port=options.port, services=[MyDateService(),])
```

Now, this code will actually work with *both* ZSI 2.0 and 2.1, as it provides a ZSI-2.1-dummy for the `@adapt`-decoration. However, as we have seen in the SquareService example, it is very feasible to manually make the changes needed for the different library versions, without depending on the “magic” `service_adaptor` provides.

Here´s the implementation of the `service_adaptor` module (you can skip this section if you're running 2.1 anyway or if you are not interested in the inner workings of this[10]):

```
$ cat service_adaptor.py
import types
# Based on "Method enhancement" Python cookbook recipee by Ken Seehof
# (http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/81982)}
class ServiceAdaptor21(object):
    """Helper class factory to provide ZSI 2.1 code compatibility.
    Takes a ZSI 2.0 service class and returns a class derived from this service
    class, that uses the ZSI 2.1 soap_method signature + return value
    conventions.
    Can be used to make a ZSI 2.1-based service class implementation work with
    the ZSI 2.0 library.
    Parameters:
        ZSI20ServiceClass: Service class, as imported from ZSI 2.0-generated
            module code
    """

    @staticmethod
    def enhanced_method(method, replacement):
        """Return a callable wrapped with a replacement callable"""
        def _f(*args, **kwargs):
            return replacement(method, *args, **kwargs)
        _f.__name__ = method.__name__
        return _f

    @staticmethod
    def _adaptTo21_soap_(orig_method, self, ps, **kwargs):
        """Wrapper method implementation, ZSI 2.0 soap_ method to ZSI 2.1 soap_
        method interface
        """
        response = orig_method(self, ps)
        return self.request, response

    @staticmethod
    def _adaptTo20_soap_(orig_method, self, ps, **kwargs):
        """Wrapper method implementation, ZSI 2.1 soap_ method to ZSI 2.0 soap_
        method interface
        """
        request, response = orig_method(self, ps)
        return response

    @staticmethod
    def adapt(method):
        """Method decorator to decorate ZSI 2.1-conforming soap_ method to ZSI
        2.0 method interface
        """
        return ServiceAdaptor21.enhanced_method(
            method, ServiceAdaptor21._adaptTo20_soap_)
```

[10]You might want to look at the Python cookbook recipee ”Method enhancement” by Ken Seehof (http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/81982) for background

```
    def __new__(cls, ZSI20ServiceClass):
        derived_dict = {}

        for methodName in ZSI20ServiceClass.__dict__:
            if methodName.startswith("soap_"):
                method = ZSI20ServiceClass.__dict__[methodName]
                derived_dict[methodName] = cls.enhanced_method(
                    method, cls._adaptTo21_soap_)

        return types.ClassType(ZSI20ServiceClass.__name__,
                               bases=(ZSI20ServiceClass,), dict=derived_dict)
```

3.3 A Python ZSI client for the DateService

3.3.1 Using generated type mapping

As seen before, we use the ZSI-generated code to write a client that calls both service methods:

ZSI 2.1

```
#! /apps/pydev/hjoukl/bin/python2.4
# *** ZSI 2.1 ***
import sys, time
from optparse import OptionParser
from ZSI.version import Version as zsiversion
print "*** ZSI version %s ***" % (zsiversion,)
from DateService_client import *

def main():
    op = OptionParser(usage="%prog [options]")
    op.add_option("-u", "--url", help="service URL", metavar="URL")
    op.add_option("-i", "--input", type="string",
                  help="input string for getCurrentDate WS method",
                  metavar="INPUT")

    options, args = op.parse_args()

    loc = simple_Date_ServiceLocator()

    service = loc.getDateService_Port(url=options.url, tracefile=sys.stdout)

    while 1:
        offset = raw_input("Enter offset as int [0]: ")
        try:
            offset = int(offset)
        except ValueError:
            offset = 0

        currentRequest = getCurrentDateRequest()
        currentRequest._input = options.input
        currentResponse = service.getCurrentDate(currentRequest)

        # We use the current date as input to getDate
        dateRequest = getDateRequest(offset=offset, some-
day=currentResponse._today)
        response = service.getDate(dateRequest)

        print '\n\nRESULT'
        print '%10s = %s' % ('today', make_asctime(currentResponse._today))
        print '%6s + %d = %s' % ('to-
day', dateRequest._offset, make_asctime(response._day))


# just a helper
def make_asctime(date_object):
    timeTuple = (date_object._year, date_object._month, date_object._day,
                 date_object._hour, date_object._minute, date_object._second,
                 date_object._weekday, date_object._dayOfYear, date_object._dst
                 )
    return time.asctime(timeTuple)


if __name__ == '__main__':
    main()
```

As can be seen, ZSI provides us with the `getCurrentDateRequest` and `getDateRequest` classes. These handle the serialization transparently and we use them to set the argument values. Then, we hand them into the corresponding methods of the service port object retrieved with the `simple_Date_ServiceLocator-.getDateService_Port()`method.

Running this client gives the following output:

```
$ ./myDateClient.py -u "http://adevp02:8080/DateService?wsdl" -i HALLO
*** ZSI version (2, 1, 0) ***
Enter offset as int [0]: 0
_________________________________ Thu Jan  3 08:40:56 2008 REQUEST:
<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:ns1="urn:DateService.wsdl">
<ns1:getCurrentDate>
<input>HALLO</input>
</ns1:getCurrentDate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
_________________________________ Thu Jan  3 08:40:56 2008 RESPONSE:
200
OK
-------
Server: ZSI/1.1 BaseHTTP/0.3 Python/2.4.4
Date: Thu, 03 Jan 2008 07:40:56 GMT
Content-type: text/xml; charset="utf-8"
Content-Length: 627

<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:ns1="urn:DateService.wsdl">
<ns1:getCurrentDateResponse>
<today>
<year>2008</year><month>1</month><day>3</day>
```

```
<hour>8</hour><minute>40</minute><second>56</second>
<weekday>3</weekday><dayOfYear>3</dayOfYear><dst>0</dst>
</today>
</ns1:getCurrentDateResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
_________________________________ Thu Jan  3 08:40:56 2008 REQUEST:
<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:ns1="urn:DateService.wsdl">
<ns1:getDate>
<offset>0</offset>
<someday>
<year>2008</year><month>1</month><day>3</day>
<hour>8</hour><minute>40</minute><second>56</second>
<weekday>3</weekday><dayOfYear>3</dayOfYear><dst>0</dst>
</someday>
</ns1:getDate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
_________________________________ Thu Jan  3 08:40:56 2008 RESPONSE:
200
OK
-------
Server: ZSI/1.1 BaseHTTP/0.3 Python/2.4.4
Date: Thu, 03 Jan 2008 07:40:56 GMT
Content-type: text/xml; charset="utf-8"
Content-Length: 609

<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:ns1="urn:DateService.wsdl">
<ns1:getDateResponse>
<day>
<year>2008</year><month>1</month><day>3</day>
<hour>8</hour><minute>40</minute><second>56</second>
<weekday>3</weekday><dayOfYear>3</dayOfYear><dst>0</dst>
</day>
</ns1:getDateResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>


RESULT
     today = Thu Jan  3 08:40:56 2008
 today + 0 = Thu Jan  3 08:40:56 2008
Enter offset as int [0]: ^C
```

Note that we gave an explicit URL of “http://adevp02:8080/DateService?wsdl”. ZSI transparently uses the relevant part of this URL for service invocation, silently ignoring the “?wsdl” suffix.

ZSI 2.0

```
#! /apps/pydev/bin/python2.4
# *** ZSI 2.0 ***
import sys, time
from copy import copy
from optparse import OptionParser
from ZSI.version import Version as zsiversion
print "*** ZSI version %s ***" % (zsiversion,)
from DateService_services import *
def main():
    op = OptionParser(usage="%prog [options]")
    op.add_option("-u", "--url", help="service URL", metavar="URL")
    op.add_option("-i", "--input", type="string",
                  help="input string for getCurrentDate WS method",
                  metavar="INPUT")

    options, args = op.parse_args()

    loc = simple_Date_ServiceLocator()

    service = loc.getDateService_PortType(url=options.url, tracefile=sys.stdout)

    while 1:
        offset = raw_input("Enter offset as int [0]: ")
        try:
            offset = int(offset)
        except ValueError:
            offset = 0

        currentRequest = getCurrentDateRequest()
        currentRequest._input = options.input
        currentResponse = service.getCurrentDate(currentRequest)

        # We use the current date as input to getDate
        dateRequest = getDateRequest()
        dateRequest._offset = offset

        # Workaround for ZSI 2.0 bug #1755740:
        # Multiple calls to method _get_type_or_substitute erroneously changes
        # attributes of a substitute typecode without making a (shallow) copy
        currentResponse._today.typecode = copy(currentResponse._today.typecode)

        dateRequest._someday = currentResponse._today
        response = service.getDate(dateRequest)

        print '\n\nRESULT'
        print '%10s = %s' % ('today', make_asctime(currentResponse._today))
        print '%6s + %d = %s' % ('to-
day', dateRequest._offset, make_asctime(response._day))


# just a helper
def make_asctime(date_object):
    timeTuple = (date_object._year, date_object._month, date_object._day,
                 date_object._hour, date_object._minute, date_object._second,
                 date_object._weekday, date_object._dayOfYear, date_object._dst
                 )
    return time.asctime(timeTuple)
```

```
if __name__ == '__main__':
    main()
```

Again the ZSI 2.0 implementation differs only in details:

- import the generated code from `DateService_services` (rather than `DateService_client` in ZSI 2.1)
- retrieve the service binding through the `simple_Date_ServiceLocator.getDateService_PortType()` method (rather than `simple_Date_ServiceLocator.getDateService_Port()` in ZSI 2.1
- instantiate the `getDateRequest` structure and then populate it (as opposed to using keyword args in ZSI 2.1)

Unfortunately, ZSI 2.0 has a bug that bites us so we need to use a workaround here (see code comments).

3.3.2 Using a ServiceProxy client

Alternatively, we can implement a client with the ServiceProxy class without using the generated code. This version works for both ZSI 2.0 & 2.1:

```
# *** ZSI 2.0 & 2.1 ***
import sys, time
from optparse import OptionParser
from ZSI.version import Version as zsiversion
print "*** ZSI version %s ***" % (zsiversion,)
from ZSI.ServiceProxy import ServiceProxy
op = OptionParser(usage="%prog [options]")
op.add_option("-u", "--url", help="service URL (mandatory)", metavar="URL")
op.add_option("-i", "--input", type="string",
              help="input string for getCurrentDate WS method",
              metavar="INPUT")
# just a helper
def make_asctime(date_dict):
    timeTuple = (date_dict["year"], date_dict["month"], date_dict["day"],
                 date_dict["hour"], date_dict["minute"], date_dict["second"],
                 date_dict["weekday"], date_dict["dayOfYear"],
                 date_dict["dst"])
    return time.asctime(timeTuple)
options, args = op.parse_args()
if not options.url:
    op.print_help()
    sys.exit(1)
else:
    # "Canonicalize" WSDL URL
    options.url = options.url.replace("?WSDL", "?wsdl")
    if not options.url.endswith("?wsdl"):
        options.url += "?wsdl"

    service = ServiceProxy(wsdl=options.url, tracefile=sys.stdout)

    print '\nAccessing service DateService...'
    while 1:
        offset = raw_input("Enter offset as int [0]: ")
        try:
            offset = int(offset)
        except ValueError:
            offset = 0

        currentResponse = service.getCurrentDate(input=options.input)
        # We get back a dictionaray
        print "currentResponse:", currentResponse

        # Manually fill the argument dict for demonstration purposes only
        somedayDict = {
            'year': currentResponse['today']['year'],
            'month': currentResponse['today']['month'],
            'day': currentResponse['today']['day'],
            'hour': currentResponse['today']['hour'],
            'minute': currentResponse['today']['minute'],
            'second': currentResponse['today']['second'],
            'weekday': currentResponse['today']['weekday'],
            'dayOfYear': currentResponse['today']['dayOfYear'],
            'dst': currentResponse['today']['dst'],
        }

        # We use the current date as input to getDate
        response = service.getDate(offset=offset,
                                   someday=currentResponse["today"])
        responseFromDict = service.getDate(offset=offset, someday=somedayDict)

        print '\n\nRESULT'
        print '%10s = %s' % ('today', make_asctime(currentResponse["today"]))
        print '%6s + %d = %s' % ('today', offset, make_asctime(response["day"]))
```

This is the output of a sample client sesssion of the ServiceProxy solution:

```
$ /apps/pydev/bin/python2.4 ./myDateSPclient.py -
u "http://adevp02:8080/DateService?wsdl" -i HALLO
*** ZSI version (2, 0, 0) ***
Accessing service DateService...
Enter offset as int [0]: 1
_________________________________ Wed Jan  2 17:46:50 2008 REQUEST:
<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:ns1="urn:DateService.wsdl">
<ns1:getCurrentDate>
<input xsi:type="xsd:string">HALLO</input>
</ns1:getCurrentDate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
_________________________________ Wed Jan  2 17:46:50 2008 RESPONSE:
200
OK
-------
Server: ZSI/1.1 BaseHTTP/0.3 Python/2.4.4
Date: Wed, 02 Jan 2008 16:46:50 GMT
Content-type: text/xml; charset="utf-8"
Content-Length: 628
<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header>
```

```
<SOAP-ENV:Body xmlns:ns1="urn:DateService.wsdl">
<ns1:getCurrentDateResponse>
<today>
<year>2008</year>
<month>1</month>
<day>2</day><hour>17</hour><minute>46</minute><second>50</second>
<weekday>2</weekday><dayOfYear>2</dayOfYear><dst>0</dst>
</today>
</ns1:getCurrentDateResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
currentResponse: {'today': {'second': 50, 'day-
OfYear': 2, 'day': 2, 'month': 1, 'minute': 46, 'dst': 0, 'week-
day': 2, 'hour': 17, 'year': 2008}}
_________________________________ Wed Jan  2 17:46:50 2008 REQUEST:
<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:ns1="urn:DateService.wsdl">
<ns1:getDate>
<offset xsi:type="xsd:integer">1</offset>
<someday xmlns:ns2="DateType_NS" xsi:type="ns2:Date">
<year>2008</year><month>1</month><day>2</day>
<hour>17</hour><minute>46</minute><second>50</second>
<weekday>2</weekday><dayOfYear>2</dayOfYear><dst>0</dst>
</someday>
</ns1:getDate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
_________________________________ Wed Jan  2 17:46:50 2008 RESPONSE:
200
OK
-------
Server: ZSI/1.1 BaseHTTP/0.3 Python/2.4.4
Date: Wed, 02 Jan 2008 16:46:50 GMT
Content-type: text/xml; charset="utf-8"
Content-Length: 610

<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:ns1="urn:DateService.wsdl">
<ns1:getDateResponse>
<day>
<year>2008</year><month>1</month><day>3</day>
<hour>17</hour><minute>46</minute><second>50</second>
<weekday>3</weekday><dayOfYear>3</dayOfYear><dst>0</dst>
</day></ns1:getDateResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
_________________________________ Wed Jan  2 17:46:50 2008 REQUEST:
<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
```

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:ns1="urn:DateService.wsdl">
<ns1:getDate>
<offset xsi:type="xsd:integer">1</offset>
<someday xmlns:ns2="DateType_NS" xsi:type="ns2:Date">
<year>2008</year><month>1</month><day>2</day>
<hour>17</hour><minute>46</minute><second>50</second>
<weekday>2</weekday><dayOfYear>2</dayOfYear><dst>0</dst>
</someday></ns1:getDate></SOAP-ENV:Body></SOAP-ENV:Envelope>
_________________________________ Wed Jan  2 17:46:50 2008 RESPONSE:
200
OK
-------
Server: ZSI/1.1 BaseHTTP/0.3 Python/2.4.4
Date: Wed, 02 Jan 2008 16:46:50 GMT
Content-type: text/xml; charset="utf-8"
Content-Length: 610

<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header><SOAP-
ENV:Body xmlns:ns1="urn:DateService.wsdl">
<ns1:getDateResponse>
<day>
<year>2008</year><month>1</month><day>3</day>
<hour>17</hour><minute>46</minute><second>50</second>
<weekday>3</weekday><dayOfYear>3</dayOfYear><dst>0</dst></day>
</ns1:getDateResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>


RESULT
     today = Wed Jan  2 17:46:50 2008
 today + 1 = Thu Jan  3 17:46:50 2008
Enter offset as int [0]:
```

Comments:

- We have hinted so and now this becomes important: For ZSI 2.1, the presented code will only work if it is run from a directory that *does not* contain `wsdl2py`-generated code, see section 2.3.2. This is because the ServiceProxy cache code, as opposed to the `wsdl2py`-generated code, does not use "ZSI aname modification".[11] This is not a problem for ZSI 2.0 as it uses different cache module naming. We consider this a bug in ZSI 2.1alpha.
- For demonstration purposes only, the `getDate()` method gets actually invoked twice, the second time with manually created dictionary input data.

[11]So one module will e.g. expect names not to start with "_" while the other one will.

3.4 A gSOAP C++ client for the DateService

3.4.1 Code generation from WSDL

1. Header:

```
$ /apps/pydev/bin/wsdl2h -
o dateService.h http://adevp02:8080/DateService?wsdl

**  The gSOAP WSDL parser for C and C++ 1.2.9l
**  Copyright (C) 2000-2007 Robert van Engelen, Genivia Inc.
**  All Rights Reserved. This product is provided "as is", without any war-
ranty.
**  The gSOAP WSDL parser is released under one of the following two li-
censes:
**  GPL or the commercial license by Genivia Inc. Use option -
l for more info.

Saving dateService.h

Cannot open file 'typemap.dat'
Problem reading type map file typemap.dat.
Using internal type definitions for C++ instead.

Connecting to 'http://adevp02:8080/DateService?wsdl' to re-
trieve WSDL/XSD... connected, receiving...
Warning: part 'today' uses literal style and should refer to an ele-
ment rather than a type
Warning: part 'input' uses literal style and should refer to an ele-
ment rather than a type
Warning: part 'input' uses literal style and should refer to an ele-
ment rather than a type
Warning: part 'input' uses literal style and should refer to an ele-
ment rather than a type
Warning: part 'day' uses literal style and should refer to an ele-
ment rather than a type
Warning: part 'offset' uses literal style and should refer to an ele-
ment rather than a type
Warning: part 'someday' uses literal style and should refer to an ele-
ment rather than a type
Warning: part 'offset' uses literal style and should refer to an ele-
ment rather than a type
Warning: part 'someday' uses literal style and should refer to an ele-
ment rather than a type
Warning: part 'offset' uses literal style and should refer to an ele-
ment rather than a type
Warning: part 'someday' uses literal style and should refer to an ele-
ment rather than a type

To complete the process, compile with:
soapcpp2 dateService.h
```

gSOAP allows you to modify code generation e.g. with regard to schema namespace bindings (names of prefixes, ...) or type mappings. Take a look at the comments in the generated `dateService.h` for instructions.

2. Stub code:

```
$ /apps/pydev/bin/soapcpp2 -I /data/pydev/DOWNLOADS/WebServices/C++/gsoap-
2.7/soapcpp2/import dateService.h

**  The gSOAP Stub and Skeleton Compiler for C and C++ 2.7.9l
** Copyright (C) 2000-2007, Robert van Engelen, Genivia Inc.
** All Rights Reserved. This product is provided "as is", without any war-
ranty.
** The gSOAP compiler is released under one of the following three li-
censes:
** GPL, the gSOAP public license, or the commercial license by Genivia Inc.

Saving soapStub.h
Saving soapH.h
Saving soapC.cpp
Saving soapClient.cpp
Saving soapClientLib.cpp
Saving soapServer.cpp
Saving soapServerLib.cpp
Using ns3 service name: DateService_USCOREBinding
Using ns3 service style: document
Using ns3 service encoding: literal
Using ns3 service location: http://adevp02:8080/DateService
Using ns3 schema namespace: urn:DateService.wsdl
Saving soapDateService_USCOREBindingProxy.h client proxy
Saving soapDateService_USCOREBindingObject.h server object
Saving DateService_USCOREBinding.getCurrentDate.req.xml sample SOAP/XML re-
quest
Saving DateService_USCOREBinding.getCurrentDate.res.xml sample SOAP/XML re-
sponse
Saving DateService_USCOREBinding.getDate.req.xml sample SOAP/XML request
Saving DateService_USCOREBinding.getDate.res.xml sample SOAP/XML response
Saving DateService_USCOREBinding.nsmap namespace mapping table

Compilation successful
```

3.4.2 Client implementation

As already announced in section 2.4.2 we´ll now access the service a bit more elegantly through the `DateService_USCOREBinding` proxy class, which can be found in `soapDateService_USCOREBindingProxy.h`. We use it to call the two WS-methods in our sample client:

```
// Contents of file "myclient_use_proxy.cpp"
//#include "soapH.h";
#include "soapDateService_USCOREBindingProxy.h"
#include "DateService_USCOREBinding.nsmap"
int main()
{
    DateService_USCOREBinding ds;
    ns2__Date *today, *someday;
    ns3__getCurrentDateResponse today_response;
    ns3__getDateResponse someday_response;


    std::string text, input;

    text="TEST";

    std::cout << "(1) Calling 'getCurrentDate()'-
Web Service method:" << std::endl;
    if(ds.ns3__getCurrentDate(text, today_response) == SOAP_OK)
        {
            today = today_response.today;
            std::cout << "\nCurrent date:" << std::endl;
            std::cout << "\tyear: "  << *today->year  << std::endl;
            std::cout << "\tmonth: " << *today->month << std::endl;
            std::cout << "\tday: "   << *today->day   << std::endl;
            std::cout << "\thour: "   << *today->hour   << std::endl;
            std::cout << "\tminute: "   << *today->minute   << std::endl;
            std::cout << "\tsecond: "   << *today->second   << std::endl;
        }
    else
        soap_print_fault(ds.soap, stderr);


    std::cout << "\n(2) Calling 'getDate()'- Web Service method:" << std::endl;
    std::cout << "\n(2)Please enter an integer for the 'offset'"<< std::endl;
    std::cout << "\toffset = ";
    std::cin >> input;

    someday = today;
    if(ds.ns3__getDate(input, someday, someday_response) == SOAP_OK)
        {
            someday = someday_response.day;
            std::cout << "\nSome Day:" << std::endl;
            std::cout << "\tyear: " << *someday->year << std::endl;
            std::cout << "\tmonth: "<< *someday->month << std::endl;
            std::cout << "\tday: " << *someday->day << std::endl;
            std::cout << "\thour: "   << *today->hour   << std::endl;
            std::cout << "\tminute: "   << *today->minute   << std::endl;
            std::cout << "\tsecond: "   << *today->second   << std::endl;
        }
    else
    return 0;

}
```

Comments:

- You can find the classes/structs that represent the WSDL input and output datatypes in the generated file `soapStub.h`. This file is really pretty much self-explaining, so the usage in client implementations should be straightforward.

3.4.3 gSOAP Client compilation

gcc 2.95.2

```
 $ g++ -o myClient_use_proxy -I/apps/pydev/include -L/apps/pydev/lib
-R /apps/prod/lib myClient_use_proxy.cpp soapC.cpp soapClient.cpp
-lsocket -lgsoap++
```

gcc 3.4.4

```
 $ /apps/local/gcc/3.4.4/bin/g++ -o myClient_use_proxy -
R /apps/prod/gcc/3.4.4/lib -I/apps/pydev/gcc/3.4.4/include -
L/apps/pydev/gcc/3.4.4/lib soapC.cpp soapClient.cpp myClient_use_proxy.cpp -
lgsoap++ -lsocket -lnsl
```

Running the client gives the following output:

```
$ ./myClient_use_proxy
(1) Calling 'getCurrentDate()'- Web Service method:
```

```
Current date:
        year: 2008
        month: 1
        day: 8
        hour: 17
        minute: 13
        second: 30

(2) Calling 'getDate()'- Web Service method:

(2)Please enter an integer for the 'offset'
        offset = 88

Some Day:
        year: 2008
        month: 4
        day: 5
        hour: 17
        minute: 13
        second: 30
```

4 A document/literal service: The FinancialService

The previous example services had in common that they used rpc/literal binding. While this is WS-I-compliant, there is a tendency to propagate the usage of the document/literal fashion. With rpc/literal, you might define types or elements to be used in the request and response messages, in the `<types>` section. We did that for the DateService, see section 3.1. You use these elements or types as arguments or receive them as return value of the service call. They are "packed" into the request or the response tags in the actual SOAP message, as can be seen in the sample client output in 3.3.

The difference with document/literal binding is that the *complete* message inside `<SOAP-ENV:Body>` is described in the `<types>` section, as an XML schema - not only the call parameters/return values. The big advantage is obvious: Such a message can be XML Schema-validated.

As it is quite possible with document/literal to leave the service method name out of the request/response message at all, depending on your `<types>` definitions, there can also be a downside to this. Because if you do so, it might be difficult for the server implementation to dispatch correctly.[12] To avoid this, we explicitly model the method name into our request, as a toplevel element. This is called "document-wrapped" style.

Most of the steps from WSDL to implementation should be familiar by now; we will thus concentrate on the doc/literal differences and mainly present the (commented) code.

4.1 The FinancialService WSDL

The FinancialService is described in `FinancialService.wsdl`:

```
<?xml version="1.0"?>
<definitions name="FinancialService"
  targetNamespace="http://services.zsiserver.net/FinancialService.wsdl"
  xmlns:tns="http://services.zsiserver.net/FinancialService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:finType="http://services.zsiserver.net/FinancialService_NS"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://services.zsiserver.net/FinancialService_NS"
```

[12]At least if dispatching does not rely on the SOAPAction header field.

```
      elementFormDefault="qualified">
      <!-- the getPV operation request message content, in doc-wrapped manner
        (with toplevel element that explicitly names the operation) -->
      <element name="getPV">
        <complexType>
          <sequence>
            <element name="irate" type="xsd:float"/>
            <element name="CFSequence">
              <complexType>
                <sequence>
                  <element name="CF" minOccurs="0" maxOc-
curs="unbounded" type="xsd:float"/>
                </sequence>
              </complexType>
            </element>
          </sequence>
        </complexType>
      </element>

      <!-- the answer to the getPV operation request -->

      <element name="PV" type="xsd:float"/>

    </schema>
  </types>


  <message name="getPVRequest">
    <part name="msg" element="finType:getPV"/>
  </message>

  <message name="getPVResponse">
    <part name="msg" element="finType:PV"/>
  </message>


  <portType name="FinancialService_PortType">
    <operation name="getPV">
      <input message="tns:getPVRequest"/>
      <output message="tns:getPVResponse"/>
    </operation>
  </portType>


  <binding name="FinancialService_Binding" type="tns:FinancialService_PortType">
    <soap:binding style="document" trans-
port="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getPV">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>


  <service name="FinancialService">
    <documentation>Financial Web Service. Methods: -getPV(irate, CFSequence):
```

```
Return present value for given interest rate and Cash Flows.</documentation>
    <port name="FinancialService_Port" binding="tns:FinancialService_Binding">
      <soap:address loca-
tion="http://services.zsiserver.net:8080/FinancialService"/>
    </port>
  </service>

</definitions>
```

Notes:

- The FinancialService´s getPV operation returns the net present value for a given interest rate *r* and a series of cash flows. It uses document/literal binding, so the full ingoing and outgoing message structure is defined in the WSDL `<types>` XML Schema section.
- The `soapAction` attribute is empty (`<soap:operation soapAction=""/>`). This means the server will have to use a different way to dispatch to the method implementation.
- Note the use of `elementFormDefault=■qualified■` in the XML Schema root element. This basically means that local elements (you might say sub-elements) must be namespace-qualified. The XML Schema default for this attribute is "unqualified" but we ran into interoperability problems when not setting this to "qualified", which seems to be a ZSI 2.0/2.1alpha bug. *Note: By the time of this writing, this has already been fixed with ZSI svn revision r1440.*

4.2 A Python ZSI FinancialService server

4.2.1 Code generation from WSDL

We now take advantage of ZSI´s `--complexType` option for code generation, to get some convenience getters, setters and factories:

ZSI 2.1

```
/apps/pydev/hjoukl/bin/wsdl2py --complexType FinancialService.wsdl
==> FinancialService_client.py
==> FinancialService_server.py
==> FinancialService_types.py
```

ZSI 2.0

1. `wsdl2py`:

   ```
   /apps/pydev/bin/wsdl2py --complexType -f FinancialService.wsdl
   ==> FinancialService_services.py
   ==> FinancialService_services_types.py
   ```

2. `wsdl2dispatch`:

   ```
   /apps/pydev/bin/wsdl2dispatch -f FinancialService.wsdl
   ==> FinancialService_services_server.py
   ```

4.2.2 The FinancialService server implementation

ZSI 2.0 & 2.1 Following the DateService example (3.2), we create a server implementation that runs with both ZSI version 2.0 and 2.1:

```
from optparse import OptionParser
# Import the ZSI stuff you'd need no matter what
from ZSI.wstools import logging
from ZSI import ServiceContainer
from ZSI.version import Version as zsiversion
print "*** ZSI version %s ***" % (zsiversion,)
# Import the generated Server Object
try:
    # 2.1alpha
    from FinancialService_server import *
    # Dummy implementation
    def adapt(method):
        return method
except ImportError, e:
    # 2.0final
    from FinancialService_services_server import *
    from service_adaptor import ServiceAdaptor21
    adapt = ServiceAdaptor21.adapt
    # Wrap generated 2.0 class to 2.1 soap_ method behaviour
    FinancialService = ServiceAdaptor21(FinancialService)
# Create a Server implementation by inheritance
# 2.1alpha implementation
class MyFinancialService(FinancialService):

    # Make WSDL available for HTTP GET
    _wsdl = "".join(open("FinancialService.wsdl").readlines())

    @adapt
    def soap_getPV(self, ps, **kw):
        # Call the generated base class method to get appropriate
        # input/output data structures
        request, response = FinancialService.soap_getPV(self, ps, **kw)
        # Comment that out if need be to see what API these objects offer:
        #print help(request)
        #print help(response)

        # Using getters/setters
        irate = request.get_element_irate()
        CFs = request.get_element_CFSequence().get_element_CF()

        # Alternative way to access the input data
        #irate = request._irate
        #CFs = request._CFSequence._CF

        # Invoke the service worker code
        PV = self.getPV(irate, CFs)
        #print "PV:", PV

        # assign return value to response object
        response = getPVResponse(PV)

        # Another way to create a serializable response object
        #class SimpleTypeWrapper(float):
        #    typecode = response.typecode
        #response = SimpleTypeWrapper(PV)

        return request, response

    def getPV(self, irate, cashFlows):
        # worker code method
        t = 0
        PV = 0.0
        for CF in cashFlows:
            PV += (CF or 0.0) * ((irate / 100.0 + 1) ** (-t))
            t += 1
        return PV


op = OptionParser(usage="%prog [options]")
op.add_option("-l", "--loglevel", help="loglevel (DEBUG, WARN)",
              metavar="LOGLEVEL")
op.add_option("-p", "--port", help="HTTP port",
              metavar="PORT", default=8080, type="int")
options, args = op.parse_args()

# set the loglevel according to cmd line arg
if options.loglevel:
    loglevel = eval(options.loglevel, logging.__dict__)
    logger = logging.getLogger("")
    logger.setLevel(loglevel)

# Run the server with a given list of services
ServiceContainer.AsServer(port=options.port, services=[MyFinancialService(),])
```

Comments:

- The service input data is now retrieved from the `request` object with `request.get_element_<name>()` getter methods. An alternative is presented in the code comments.
- If you need information on an objects attributes and can not find it in the docs, it can be quite helpful to make use of the power of introspection, and of docstrings, e.g. putting things like

  ```
  print help(request)
  print help(response)
  ```

  into the server code.
- Remember we left out the `soapACTION` field from the WSDL file (we left it empty, to be precise). So the dispatch to a service´s service method must now happen another way. And indeed, ZSI is capable of dispatching by making use of the name of the toplevel in-message element name. That´s the benefit of following the "document-wrapped" style, see above.

4.3 A Python ZSI client for the FinancialService

Making use of the generated stub code, this is a client implementation that works with both ZSI 2.0 & 2.1:

```
import sys, time
from optparse import OptionParser
from ZSI.version import Version as zsiversion
print "*** ZSI version %s ***" % (zsiversion,)
try:
    # 2.1alpha
    from FinancialService_client import *
    portMethodName = "getFinancialService_Port"
except ImportError, e:
    # 2.0final
    from FinancialService_services import *
    portMethodName = "getFinancialService_PortType"
def main():
    op = OptionParser(usage="%prog [options]")
    op.add_option("-u", "--url", help="service URL", metavar="URL")
```

```
    options, args = op.parse_args()
    loc = FinancialServiceLocator()

    service = getattr(loc, portMethodName)(url=options.url, trace-
file=sys.stdout)


    print '\nAccessing service FinancialService, method getPV...'

    while 1:
        irate = None
        while not (0.0 <= irate <= 100.0):
            try:
                irate = float(raw_input("Enter interest rate in percent: "))
            except ValueError, e:
                print e
        CFs = []
        period = 0
        while 1:
            try:
                CFs.append(float(raw_input("Enter CF(t=%s) [Ctrl-
C to end]: " % (period,))))
                period += 1
            except ValueError, e:
                print e
            except KeyboardInterrupt:
                print "...done."
                break

        #print "CFs list is:", CFs
        #print "Calculation interest rate is:", irate

        getPV = getPVRequest()

        getPV.set_element_irate(irate)
        CFSequence = getPV.new_CFSequence()
        CFSequence.set_element_CF(CFs)
        getPV.set_element_CFSequence(CFSequence)

        # Alternatively, a valid request data structure can be provided like
        # this (e.g. you do not want to use --complexType getters/setters)
        #class CFSequence_class:
        #    _CF = CFs
        #getPV._irate = irate
        #getPV._CFSequence = CFSequence_class

        result = service.getPV(getPV)
        print 'result:', result


if __name__ == '__main__':
    main()
```

After the necessary data has been read from the user, the appropriate attributes of the `getPVRequest` instance are set. This is done by heavy use of the getters, setters and factory methods ZSI adds when using `--complexType` code generation.

An alternative is to model the sequence of cash flow elements with a the local class `CFSequence_class`, see code comments. This class is “structurally equivalent” to the XML Schema `CFSequence` complexType as defined in the WSDL. ZSI then handles all the type mapping and serialization issues for us.

Sample client session output:

```
$ /apps/pydev/hjoukl/bin/python2.4 ./myFinancialClient.py -
u "http://adevp02:8080/FinancialService"
*** ZSI version (2, 1, 0) ***

Accessing service FinancialService, method getPV...
Enter interest rate in percent: 7
Enter CF(t=0) [Ctrl-C to end]: -100
Enter CF(t=1) [Ctrl-C to end]: 8
Enter CF(t=2) [Ctrl-C to end]: 8
Enter CF(t=3) [Ctrl-C to end]: 108
Enter CF(t=4) [Ctrl-C to end]: ^C...done.
_________________________________ Thu Jan 10 13:07:38 2008 REQUEST:
<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:ns1="http://services.zsiserver.net/FinancialService_NS">
<ns1:getPV>
<ns1:irate>7.000000</ns1:irate>
<ns1:CFSequence>
<ns1:CF>-100.000000</ns1:CF>
<ns1:CF>8.000000</ns1:CF>
<ns1:CF>8.000000</ns1:CF>
<ns1:CF>108.000000</ns1:CF>
</ns1:CFSequence>
</ns1:getPV>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
_________________________________ Thu Jan 10 13:07:38 2008 RESPONSE:
200
OK
-------
Server: ZSI/1.1 BaseHTTP/0.3 Python/2.4.4
Date: Thu, 10 Jan 2008 12:07:38 GMT
Content-type: text/xml; charset="utf-8"
Content-Length: 456

<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header>
</SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:ns1="http://services.zsiserver.net/FinancialService_NS">
<ns1:PV>2.624316</ns1:PV>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
result: 2.624316
Enter interest rate in percent:
```

4.4 A gSOAP C++ client for the FinancialService

4.4.1 Code generation from WSDL

1. Header:

```
$ /apps/pydev/bin/wsdl2h -
o financialService.h http://adevp02:8080/FinancialService?wsdl

**  The gSOAP WSDL parser for C and C++ 1.2.9l
**  Copyright (C) 2000-2007 Robert van Engelen, Genivia Inc.
**  All Rights Reserved. This product is provided "as is", without any war-
ranty.
**  The gSOAP WSDL parser is released under one of the following two li-
censes:
**  GPL or the commercial license by Genivia Inc. Use option -
l for more info.

Saving financialService.h

Cannot open file 'typemap.dat'
Problem reading type map file typemap.dat.
Using internal type definitions for C++ instead.

Connecting to 'http://adevp02:8080/FinancialService?wsdl' to re-
trieve WSDL/XSD... connected, receiving...

To complete the process, compile with:
soapcpp2 financialService.h
```

2. Stub code:

```
$ /apps/pydev/bin/soapcpp2 -I /data/pydev/DOWNLOADS/WebServices/C++/gsoap-
2.7/soapcpp2/import financialService.h

**  The gSOAP Stub and Skeleton Compiler for C and C++ 2.7.9l
**  Copyright (C) 2000-2007, Robert van Engelen, Genivia Inc.
**  All Rights Reserved. This product is provided "as is", without any war-
ranty.
**  The gSOAP compiler is released under one of the following three li-
censes:
**  GPL, the gSOAP public license, or the commercial license by Genivia Inc.

Saving soapStub.h
Saving soapH.h
Saving soapC.cpp
Saving soapClient.cpp
Saving soapClientLib.cpp
Saving soapServer.cpp
Saving soapServerLib.cpp
Using ns1 service name: FinancialService_USCOREBinding
Using ns1 service style: document
Using ns1 service encoding: literal
Using ns1 service location: http://adevp02:8080/FinancialService
Using ns1 schema names-
pace: http://services.zsiserver.net/FinancialService.wsdl
Saving soapFinancialService_USCOREBindingProxy.h client proxy
Saving soapFinancialService_USCOREBindingObject.h server object
Saving FinancialService_USCOREBinding.getPV.req.xml sample SOAP/XML request
Saving FinancialService_USCOREBinding.getPV.res.xml sample SOAP/XML response
Saving FinancialService_USCOREBinding.nsmap namespace mapping table

Compilation successful
```

4.4.2 Client implementation

```
// Contents of file "myclient_use_proxy.cpp"
```

```
#include "soapFinancialService_USCOREBindingProxy.h"
#include "FinancialService_USCOREBinding.nsmap"

int main()
{
    FinancialService_USCOREBinding fs;
    _ns2__getPV getPV;

    float PV;
    float irate;
    float CFval;

    std::string input;


    while (1) {
        std::cout << "Enter interest rate in %: ";
        std::cin >> irate;

        std::vector<float> CF;
        for (int t=0;; t++) {
            std::cout << "Enter CF(t=" << t << ")[e to exit]: ";
            std::cin >> input;
            if (input != "e") {
                CFval = atof(input.c_str());
                CF.push_back(CFval);
            } else {
                break;
            }
        }

        getPV.irate = irate;
        getPV.CFSequence.CF = CF;

        if (fs.__ns1__getPV(&getPV, PV) == SOAP_OK) {
            std::cout << "PV=" << PV << std::endl;
        } else
            soap_print_fault(fs.soap, stderr);
    }
}
```

4.4.3 gSOAP Client compilation

gcc 2.95.2

```
g++ -o myClient -R/apps/prod/lib  -I/apps/pydev/include -
L/apps/pydev/lib soapC.cpp soapClient.cpp myClient_use_proxy.cpp -lgsoap++ -
lsocket
```

gcc 3.4.4

```
/apps/local/gcc/3.4.4//bin/g++ -o myClient -R/apps/prod/gcc/3.4.4/lib  -
I/apps/pydev/gcc/3.4.4/include -
L/apps/pydev/gcc/3.4.4/lib soapC.cpp soapClient.cpp myClient_use_proxy.cpp -
lgsoap++ -lsocket -lnsl
```

5 Afterthoughts

With ZSI 2.0 and even more so the upcoming 2.1 version, ZSI has made important steps towards better usability, performance and functionality. When the original example code base presented here was developed with ZSI 1.6.1, several patches were necessary to get everything working, and a custom request handler was used to implement non-SOAPAction-based service dispatch. This time, ZSI 2.0 and 2.1 handled everything out-of-the-box, with little odds and ends. Still on the wish-list is more verbose built-in tracing capabilities; for the time being, it is necessary to either modify ZSI library code or to use some network sniffer to peek at the actual XML workload (or even the full HTTP header & body).

A weaker point of ZSI still is documentation, though some nice 3rd party resources are available these days, notably [8] and [9]. Hopefully, this is a valuable addition to the list.

A VistaSource Applixware spreadsheets as DateService clients

While Applixware 4.43 does not come with built-in web services support it is extensible with C shared libraries and its ELF macro extension language. We can use this feature to wrap gSOAP client code, making web services accessible from within Applix spreadsheets.

Note: Example compiled with gcc 2.95.2. A short try with gcc 3.4.4 showed some compilation errors which weren´t

A.0.1 Code generation from WSDL

The C/C++ client stubs must be generated as described in section3.4.1.

A.0.2 Applix client implementation

ELF C extension To make the DateService callable from Applix we need to write an ELF shared library C extension. The `ax_DateService.cpp` extension implements the gSOAP web services access:

```
#include "elfapi.h"
#include "soapDateService_USCOREBindingProxy.h"
#include "DateService_USCOREBinding.nsmap"
#define TRUE 1
extern "C" elfData getCurrentDate();
extern "C" elfData getCurrentDate_2();
extern "C" elfData getDate();
extern "C" elfData getDate_2();
/*
*       Define the function table
*/
AxCallInfo_t    funcTable[]={
        {       "Web Services", /* func type ... "financial", "math" ... */
                getCurrentDate,           /* The C routine to call  */
                "getCurrentDate",         /* name to use in Applixware (usu-
ally identical to the function name)   */
                "string getCurrentDate(string)",  /* shows the func-
tion name and its arguments */                TRUE            /* An inte-
ger that governs the treatment of ERROR and NA val-
ues,                                  and whether the function is dis-
played in the Spreadsheets Functions dialog box. */
        },
        {       "Web Services", /* func type ... "financial", "math" ... */
                getCurrentDate_2,           /* The C routine to call  */
                "getCurrentDate_2",         /* name to use in Applixware (usu-
ally identical to the function name)   */
                "date getCurrentDate_2(string)",  /* shows the func-
tion name and its arguments */                TRUE            /* An inte-
ger that governs the treatment of ERROR and NA val-
ues,                                  and whether the function is dis-
played in the Spreadsheets Functions dialog box. */
        },
        {       "Web Services", /* func type ... "financial", "math" ... */
                getDate,                  /* The C routine to call  */
                "getDate",         /* name to use in Applixware (usually identi-
cal to the function name)   */
                "date_string getDate(int, date)",   /* shows the func-
tion name and its arguments */
                TRUE            /* An integer that governs the treatment of ER-
ROR and NA values,                                  and whether the func-
tion is displayed in the Spreadsheets Functions dialog box. */
```

```
        },
        {       "Web Services", /* func type ... "financial", "math" ... */
                getDate_2,                  /* The C routine to call  */
                "getDate_2",         /* name to use in Applixware (usually iden-
tical to the function name)   */
                "date getDate_2(int, date)",   /* shows the func-
tion name and its arguments */
                TRUE            /* An integer that governs the treatment of ER-
ROR and NA values,                                    and whether the func-
tion is displayed in the Spreadsheets Functions dialog box. */
        },
        {       NULL,
                NULL,
                NULL,
                NULL,
                NULL
        }
};
/*
*       Function AxGetCallInfo returns the function table.
*       This function is called by Applixware when you run the macro
*       RPC_CONNECT@ or INSTALL_C_LIBRARY@. This function must exist
*       in the RPC program or Shared Library.
*/
DLL_EXPORT AxCallInfo_t *AxGetCallInfo()
{
        return(funcTable);
}
/*=========================================*/
/* func                                    */
/*=========================================*/
extern "C" elfData getCurrentDate(elfData args)     /* args is an array of argu-
ments passed from ELF */
{
    elfData arrayElem;
    elfData retValue;
    char *val;
    /* Check argument number & types */
    if (AxArraySize(args) != 1) {
        AxError(1, "function takes one single argument", "getCurrentDate");
    }
    arrayElem = AxArrayElement(args, 0);
    if (!AxIsString(arrayElem)) AxError(1, "argument must be a string", "getCur-
rentDate");
    DateService_USCOREBinding ds;
    ns2__Date *today;
    ns3__getCurrentDateResponse today_response;
    val = AxStrFromDataPtr(arrayElem);
    std::string text(val);
    std::string soapError = "SOAP Error";
    std::string result = "";
    if(ds.ns3__getCurrentDate(text, today_response) == SOAP_OK)
        {
            today = today_response.today;
            result = *today->year
                + "/" + *today->month
                + "/" + *today->day
                + " " + *today->hour
                + ":" + *today->minute
                + ":" + *today->second;
         }
    else
        {
            //soap_print_fault(ds.soap, stderr);  // gSOAP error
            soapError = "SOAP error";
            AxError(1, soapError.c_str(), "getCurrentDate");
            //AxError(1, "SOAP error", "getCurrentDate");
        }
     retValue = AxMakeStrData(result.length(), result.c_str());
     return retValue;
}
extern "C" elfData getCurrentDate_2(elfData args)     /* args is an array of ar-
guments passed from ELF */
{
    elfData arrayElem;
    elfData retArray;
    char *val;
    /* Check argument number & types */
    if (AxArraySize(args) != 1) {
        AxError(1, "function takes one single argument", "getCurrentDate");
    }
    arrayElem = AxArrayElement(args, 0);
    if (!AxIsString(arrayElem)) AxError(1, "argument must be a string", "getCur-
rentDate");
    DateService_USCOREBinding ds;
    ns2__Date *today;
    ns3__getCurrentDateResponse today_response;
    val = AxStrFromDataPtr(arrayElem);
    std::string text(val);
    std::string soapError = "";
    if(ds.ns3__getCurrentDate(text, today_response) == SOAP_OK)
        {
            today = today_response.today;
            retArray = AxMakeArray(0);
            retArray = AxAddIntToArray(retArray, 0, atoi((*today-
>year).c_str()));
            retArray = AxAddIntToArray(retArray, 1, atoi((*today-
>month).c_str()));
            retArray = AxAddIntToArray(retArray, 2, atoi((*today-
>day).c_str()));
            retArray = AxAddIntToArray(retArray, 3, atoi((*today-
>hour).c_str()));
            retArray = AxAddIntToArray(retArray, 4, atoi((*today-
>minute).c_str()));
            retArray = AxAddIntToArray(retArray, 5, atoi((*today-
>second).c_str()));
            retArray = AxAddIntToArray(retArray, 6, atoi((*today-
>weekday).c_str()));
            retArray = AxAddIntToArray(retArray, 7, atoi((*today-
>dayOfYear).c_str()));
            retArray = AxAddIntToArray(retArray, 8, atoi((*today-
>dst).c_str()));
         }
    else
        {
            //soap_print_fault(ds.soap, stderr);  // gSOAP error
            soapError = "SOAP error";
            AxError(1, soapError.c_str(), "getCurrentDate");
            //AxError(1, "SOAP error", "getCurrentDate");
        }
    return retArray;
}
```

```
extern "C" elfData getDate(elfData args)     /* args is an array of argu-
ments passed from ELF */ {
    elfData elf_offset, elf_date, retValue;
    /* Check argument number & types */
    if (AxArraySize(args) != 2) {
        AxError(1, "function takes 2 arguments", "getDate");
    }
    elf_offset = AxArrayElement(args, 0);
    if (!AxIsInt(elf_offset)) AxError(1, "argument must be an integer", "get-
Date");
    elf_date = AxArrayElement(args, 1);
    if (!AxIsArray(elf_date)) AxError(1, "argument must be an array", "get-
Date");
    DateService_USCOREBinding ds;
    //ns2__Date someday, date;
    ns2__Date *someday, date;
    ns3__getDateResponse someday_response;
    std::string offset(AxStrFromDataPtr(elf_offset));
    std::string year(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 0), 0)));
    std::string month(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 1), 0)));
    std::string day(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 2), 0)));
    std::string hour(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 3), 0)));
    std::string minute(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 4), 0)));
    std::string sec-
ond(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 5), 0)));
    std::string week-
day(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 6), 0)));
    std::string day-
OfYear(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 7), 0)));
    std::string dst(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 8), 0)));
    date.year = &year;
    date.month = &month;
    date.day = &day;
    date.hour = &hour;
    date.minute = &minute;
    date.second = &second;
    date.weekday = &weekday;
    date.dayOfYear = &dayOfYear;
    date.dst = &dst;
    std::string result = "";
    //retValue = AxMakeStrData(offset.length(), offset.c_str());
    //return retValue;
    //std::cout << "(1) Calling 'getCurrentDate()'-
Web Service method:" << std::endl;
    if(ds.ns3__getDate(offset, &date, someday_response) == SOAP_OK)
        {
            someday = someday_response.day;
            //someday = *(someday_response.o_USCOREsomeday);
//             std::cout << "\nCurrent date:" << std::endl;
//             std::cout << "\tyear: "  << *someday->year  << std::endl;
//             std::cout << "\tmonth: " << *someday->month << std::endl;
//             std::cout << "\tday: "   << *someday->day   << std::endl;
            result = *someday->year
                + "/" + *someday->month
                + "/" + *someday->day
                + " " + *someday->hour
                + ":" + *someday->minute
                + ":" + *someday->second;
          }
    else
        {
            //soap_print_fault(ds.soap, stderr);  // gSOAP error
            AxError(1, "SOAP error", "getDate");
        }
    retValue = AxMakeStrData(result.length(), result.c_str());
    return retValue;
    //    return retArray;
}
extern "C" elfData getDate_2(elfData args)     /* args is an array of argu-
ments passed from ELF */
{
    elfData elf_offset, elf_date, retArray;
    /* Check argument number & types */
    if (AxArraySize(args) != 2) {
        AxError(1, "function takes 2 arguments", "getCurrentDate");
    }
    elf_offset = AxArrayElement(args, 0);
    if (!AxIsInt(elf_offset)) AxError(1, "argument must be an integer", "get-
Date");
    elf_date = AxArrayElement(args, 1);
    if (!AxIsArray(elf_date)) AxError(1, "argument must be an array", "get-
Date");
    DateService_USCOREBinding ds;
    //ns2__Date someday, date;
    ns2__Date *someday, date;
    ns3__getDateResponse someday_response;
    std::string offset(AxStrFromDataPtr(elf_offset));
    std::string year(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 0), 0)));
    std::string month(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 1), 0)));
    std::string day(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 2), 0)));
    std::string hour(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 3), 0)));
    std::string minute(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 4), 0)));
    std::string sec-
ond(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 5), 0)));
    std::string week-
day(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 6), 0)));
    std::string day-
OfYear(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 7), 0)));
    std::string dst(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 8), 0)));
    date.year = &year;
    date.month = &month;
    date.day = &day;
    date.hour = &hour;
    date.minute = &minute;
    date.second = &second;
    date.weekday = &weekday;
    date.dayOfYear = &dayOfYear;
    date.dst = &dst;
    std::string result = "";
    int return_integer = TRUE;
    //std::cout << "(1) Calling 'getCurrentDate()'-
Web Service method:" << std::endl;
    if(ds.ns3__getDate(offset, &date, someday_response) == SOAP_OK)
        {
            someday = someday_response.day;
            //someday = *(someday_response.o_USCOREsomeday);
//             std::cout << "\nCurrent date:" << std::endl;
//             std::cout << "\tyear: "  << *someday->year  << std::endl;
//             std::cout << "\tmonth: " << *someday->month << std::endl;
//             std::cout << "\tday: "   << *someday->day   << std::endl;
            retArray = AxMakeArray(0);
            //std::string val;
            //val = *someday->year;
            //elfData err;
            //err = AxMakeStrData(val.length(), val.c_str());
            //AxError(1, val.c_str(), err);

            if (return_integer == TRUE)
                {
                    retArray = AxAddIntToArray(retArray, 0, atoi((*someday-
>year).c_str()));
                    retArray = AxAddIntToArray(retArray, 1, atoi((*someday-
>month).c_str()));
                    retArray = AxAddIntToArray(retArray, 2, atoi((*someday-
>day).c_str()));
                    retArray = AxAddIntToArray(retArray, 3, atoi((*someday-
>hour).c_str()));
                    retArray = AxAddIntToArray(retArray, 4, atoi((*someday-
>minute).c_str()));
                    retArray = AxAddIntToArray(retArray, 5, atoi((*someday-
>second).c_str()));
                    retArray = AxAddIntToArray(retArray, 6, atoi((*someday-
>weekday).c_str()));
                    retArray = AxAddIntToArray(retArray, 7, atoi((*someday-
>dayOfYear).c_str()));                    retArray = AxAddIntToAr-
ray(retArray, 8, atoi((*someday->dst).c_str()));
                }
            else
                {
                    retArray = AxAddStrToArray(retArray, 0, (*someday-
>year).c_str());
                    retArray = AxAddStrToArray(retArray, 1, (*someday-
>month).c_str());
                    retArray = AxAddStrToArray(retArray, 2, (*someday-
>day).c_str());
                    retArray = AxAddStrToArray(retArray, 3, (*someday-
>hour).c_str());
                    retArray = AxAddStrToArray(retArray, 4, (*someday-
>minute).c_str());
                    retArray = AxAddStrToArray(retArray, 5, (*someday-
>second).c_str());
                    retArray = AxAddStrToArray(retArray, 6, (*someday-
>weekday).c_str());
                    retArray = AxAddStrToArray(retArray, 7, (*someday-
>dayOfYear).c_str());
                    retArray = AxAddStrToArray(retArray, 8, (*someday-
>dst).c_str());
                }


            //result = *someday->year + "/" + *someday->month + "/" + *someday-
>day;
            //result = *(someday.year) + "/" + *(someday.month) + "/" + *(some-
day.day);

         }
    else
        {
            //soap_print_fault(ds.soap, stderr);  // gSOAP error
            AxError(1, "SOAP error", "getCurrentDate");
        }

    //retValue = AxMakeStrData(result.length(), result.c_str());
```

```
        //return retValue;
        return retArray;
    }
```

The code above implements the following functions:

- getCurrentDate: Return a string containing the current date. Takes a string argument.
- getCurrentDate_2: Return the current date as an array. Takes a string argument.
- getDate: Return the input date + offset as a string. Takes an integer offset and a 9-element date array as arguments.
- getDate_2: Return input date + offset as an array. Takes an integer offset and a 9-element date array as arguments.

To compile the code, you have to add the Applix ELF headers to the include path:

```
g++ -shared -o ax_DateService.so -R/apps/prod/lib
-I/apps/prod/applix/applix/versions/4.43.1021.544.343/axdata/elf
-I/apps/pydev/include -L/apps/pydev/lib soapC.cpp soap ax_DateService.cpp
-lgsoap++ -lsocket
```

The resulting shared library `ax_DateService.so` must be loaded into Applix before it can be used. This is done with the `install_c_library@(<extension lib>)` macro:

```
#define path "/data/pydev/WebServicesPresentation/DateService_rpc_literal/Applix/"
macro load()
        install_c_library@(path++"ax_DateService.so")
endmacro
macro unload()
        unbind_c_library@(path++"ax_DateService.so")
endmacro
macro reload()
        unbind_c_library@(path++"ax_DateService.so")
      install_c_library@(path++"ax_DateService.so")
endmacro
```

After the shared library has been dynamically loaded, the functions can be used like built-in functions inside the spreadsheet. To make sensible use of the `getDate_2` function which returns the date struct as an array, you have to invoke it as an array function:[13]

1. Select an array with the appropriate number of cells (9 cells, one-dimensional in our case).
2. Enter the function call, e.g.:

   ```
   =getDate_2(offset_5, someday_5)
   ```

3. Press CTRL-SHIFT-ENTER to insert the formula as array function.

The actual extension function calls might also be wrapped into additional macro code. This makes it possible to invoke the functions from command buttons or to populate predefined, hardcoded result ranges with the results of a service call. Compared to Excel, an additional possibility for Applix is the definition of an extra argument for the output range *name* (as a string). Given the name, this range can then be filled with the result value structure, removing the need to hardcode range names in the macro code.[14]

The following macro code show some of the possibilities. Refer to the code comments for explanations:

[13]Compare to section **??**.

[14]In Applix, macros can be called from spreadsheet cells but places restrictions on modifying spreadsheet cells from within a macro.

```
#include "spsheet_.am"
/*
* Macros for Web Service getCurrentDate()
*/
/* <<<<  getCurrentDate_Macro >>>>> */
'             -->  Returning ASCII string
'             -->  called from within a cell
'             -->  providing an input parameter
Macro getCurrentDate_Macro(text)
var retStr
/* Web Service call */
retStr = getCurrentDate(text)
return(retStr)
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>*/
/* <<<<  getCurrentDate_2_Macro >>>>> */
'              --> no return value: returning ASCII string with population
'               -->  called by Buttons
'               -->  no input parameter possible
'               -->  hard-coded input-parameter'
'               -->  hard coded named range for result population
Macro getCurrentDate_2_Macro()
var format ss_cell_ cellInfo
var celladdr, cell
var retStr, text
cellAddr = ss_extract_range_info@("input_3")
cellInfo = ss_get_cell@(cellAddr[0], cellAddr[1], 0)
text=cellInfo.display_str
/* Web Service call */
retStr = getCurrentDate(text)
/* Result Population */
cellAddr = ss_extract_range_info@("today_1")
 ss_put_cell@(ss_coordinate@(cellAddr[0], cellAddr[1], 0), retStr)
'Alternativ
'cell=ss_coordinate@(cellAddr[0], cellAddr[1], 0)
'new_task@("ss_put_cell@", cell, retStr)
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>*/
/* <<<<  getCurrentDate_3_Macro >>>>> */
'              --> no return value: returning date array  with population
'               -->  called by cell
'               -->  input parameter for result population
Macro getCurrentDate_3_Macro(text, today)
'for parameter to-
day the name (string) of a named range has to be passed, not the named
' ranged object by itself --> this would pass an array to the macro, where the
'content of the named range is provided. But we need the coordi-
nates of the cells of the
'named range
var format ss_cell_ cellInfo
var celladdr, cell
var retStr, date, rowStart, rowEnd, col
var k, i
/* Web Service call */
date = getCurrentDate_2(text)
/* Result Population */
'cellAddr = ss_extract_range_info@("today")
' --> hardcoded if no parameter is allowed
cellAddr = ss_extract_range_info@(today)
' --> not possible to get named range as parameter for result because
' the content of the named range is used by APPLIX NOT THE COORDINATES
```

```
rowStart=cellAddr[1]
rowEnd=cellAddr[3]
col=cellAddr[2]
k=0
for i=rowStart to rowEnd
        cellInfo= ss_get_cell@(col, i, 0)
        'date[k]= { cellInfo.display_str+0 }
        'ss_put_cell@(ss_coordinate@(cellAddr[col], cellAddr[i], 0), date[k])
        'Alternativ
        cell=ss_coordinate@(col, i, 0)
        new_task@("ss_put_cell@", cell, date[k])
        'ss_put_cell@(col,i,date[k])
        k=k+1
next i
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>*/
/*
* Macros for Web Service getDate()
*/
/* <<<<  getDate_Macro >>>>> */
'             -->  Returning ASCII string
'             -->  called from within a cell
'             -->  providing  input parameter
Macro getDate_macro(offset, someday)
var retStr, date
date=someday
/* Web Service call */
retStr = getDate(offset, date)
return(retStr)
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>*/
/* <<<<  getDate_1_Macro >>>>> */
'                -->  Not used
'              --> no return value: returning ASCII string with population
'               -->  called by Buttons  or from within a cell
'               -->  no input parameter possible
'               -->  hard-coded input-parameter'
'               -->  hard coded named range for result population
Macro getDate_1_macro()
var format ss_cell_ cellInfo
var retStr, offset, date, rowStart, rowEnd, col
var cellAddr, k, i, cell
/* Preparing Web Service input parameter */
cellAddr = ss_extract_range_info@("offset")
cellInfo = ss_get_cell@(cellAddr[0], cellAddr[1], 0)
offset=cellInfo.display_str+0
cellAddr = ss_extract_range_info@("someday")
rowStart=cellAddr[1]
rowEnd=cellAddr[3]
col=cellAddr[2]
k=0
for i=rowStart to rowEnd
        cellInfo= ss_get_cell@(col, i, 0)
        date[k]= { cellInfo.display_str+0 }
        k=k+1
next i
/* Web Service call */
retStr = getDate(offset, date)
/* Result Population */
cellAddr = ss_extract_range_info@("result_macro1")
/* Gilt fuer Start des Makros ueber Button  */
```

```
' ss_put_cell@(ss_coordinate@(cellAddr[0], cellAddr[1], 0), retStr)
/* Gilt fuer Start des Makros in einer Zelle  */
cell=ss_coordinate@(cellAddr[0], cellAddr[1], 0)
'cell="A:E12"
new_task@("ss_put_cell@", cell, retStr)
return(retStr)
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>*/
/* <<<<  getDate_2_Macro >>>>> */
'              --> no return value: returning date array with population
'               -->  called by Buttons
'               -->  no input parameter possible
'               -->  hard-coded input-parameter'
'               -->  hard coded named range for result population
Macro getDate_2_macro()
var format ss_cell_ cellInfo
var celladdr, cell
var date, rowStart, rowEnd, col
var k, i
var  offset, someday
/* Perparing Web Service input parameter */
cellAddr = ss_extract_range_info@("offset_3")
cellInfo = ss_get_cell@(cellAddr[0], cellAddr[1], 0)
'offset=cellInfo.display_str + 0
offset=cellInfo.value
cellAddr = ss_extract_range_info@("someday_3")
rowStart=cellAddr[1]
rowEnd=cellAddr[3]
col=cellAddr[2]
k=0
for i=rowStart to rowEnd
        cellInfo = ss_get_cell@(col, i, 0)
        someday[k] = {cellInfo.value}
        'someday[k]=ss_coordinate@(col, i, 0)
        'new_task@("ss_put_cell@", cell, date[k])
        'ss_put_cell@(col,i,date[k])  --> not working
        k=k+1
next i
'DUMP_ARRAY@(someday)
/* Web Serviuce call */
date = getDate_2(offset, someday)
/* Result Population */
cellAddr = ss_extract_range_info@("day_1")
' --> not possible to get named range as parameter for result because
' the content of the named range is used by APPLIX NOT THE COORDINATES
rowStart=cellAddr[1]
rowEnd=cellAddr[3]
col=cellAddr[2]
k=0
for i=rowStart to rowEnd
        cell=ss_coordinate@(col, i, 0)
        new_task@("ss_put_cell@", cell, date[k])
        'ss_put_cell@(col,i,date[k])  --> not working
        k=k+1
next i
/*
* DUMP_ARRAY@(retStruct)
*/
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>*/
/* <<<<  getDate_3_Macro >>>>> */
```

```
'               --> no return value: returning date array with population
'                -->  called by cells
'                -->  input parameter for WebService call
'                --> named range input-parameter for result population
'
Macro getDate_3_macro(offset, someday, day)
var format ss_cell_ cellInfo
var celladdr, cell
var date, rowStart, rowEnd, col
var k, i
/* Preparation Web Service input parameter not necessary --> provided by call */
/* Web Service call */
date = getDate_2(offset, someday)
/* Result Population */
'cellAddr = ss_extract_range_info@("day")
' --> hardcoded if no parameter is allowed
cellAddr = ss_extract_range_info@(day)
' --> not possible to get named range as parameter for result because
' the content of the named range is used by APPLIX NOT THE COORDINATES
rowStart=cellAddr[1]
rowEnd=cellAddr[3]
col=cellAddr[2]
k=0
for i=rowStart to rowEnd
        cell=ss_coordinate@(col, i, 0)
        new_task@("ss_put_cell@", cell, date[k])
        'ss_put_cell@(col,i,date[k])  --> not working
        k=k+1
next i
/*
* DUMP_ARRAY@(retStruct)
*/
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>>*/
```

B ZSI tracing hacks

Warning: These are more-or-less nifty recipees to inject some functionality to ZSI with regard to logging/debugging. Not tested beyond what you see here.

B.1 Detailed method call tracing

If something goes wrong with answering a service request in a ZSI server and you do not know why this is, sometimes ZSI´s fault exception message is just not enough to get a clue. As the current ZSI versions do not allow you to output more detailed call stack information, here´s a helper module `inject_tracing.py` to inject desired functionality into ZSI:

```
import types
# Based on "Method enhancement" Python cookbook recipee by Ken Seehof
# (http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/81982)}
def enhance_method(cls, methodname, decorate):
    'replace a method with an enhancement'
    method = getattr(cls, methodname)
    _f = decorate(method)
    setattr(cls, methodname, types.MethodType(_f, None, cls))
# loop over class dict and call enhance_method() function
# for all methods to modify
def enhance_all_methods(cls, decorate):
    for methodname in cls.__dict__:
        if not methodname.startswith("__") \
           or methodname in ["__call__", "__call___"]:
            method = getattr(cls, methodname)
            if isinstance(method, types.MethodType):
                enhance_method(cls, methodname, decorate)


def traced(enter_text="-->", leave_text="<--"):
    """Configurable Function decorator.
    Outputs the text given as enter_text plus the function name and arguments
    before entering the decorated function, and the leave_text string after
    having left the decorated function.
    Example:
    >>> @traced("into", "out of")
    ... def foo(x,y):
    ...     print x,y
    ...     return x
    ...
    >>> foo(1, 2)
    into foo(args=(1, 2), kwargs={})
    1 2
    out of foo()
    1
    >>>
    """
    def traced(f):
        _f = f
        def f(*args, **kwargs):
            print enter_text, "%s(args=%s, kwargs=%s)" % (_f.__name__, args,
                                                          kwargs)
            try:
                return _f(*args, **kwargs)
            finally:
                print leave_text, "%s()" % (_f.__name__)
        f.__name__ = _f.__name__
        return f
    return traced

def enhance_module(module, decorator=traced, names=[]):
    """Wrap module class methods and callables with given decorator. Applies
    to list of names only, if given, otherwise to all detected
    methods/callables.
    """
    for objname, obj in module.__dict__.items():
        if not names or objname in names:
            if isinstance(obj, (type, types.ClassType)):
                #print obj
                enhance_all_methods(obj, decorator())
            elif callable(obj):
                #print obj
                module.__dict__[objname] = decorator()(obj)
```

The FinancialService server implementation from section 4.2 can be modified like this to make use of it:

```
from optparse import OptionParser
# Import the ZSI stuff you'd need no matter what
from ZSI.wstools import logging
from ZSI import ServiceContainer
from ZSI.version import Version as zsiversion
print "*** ZSI version %s ***" % (zsiversion,)
```

```
# Import the generated Server Object
try:
    # 2.1alpha
    from FinancialService_server import *
    # Dummy implementation
    def adapt(method):
        return method
except ImportError, e:
    # 2.0final
    from FinancialService_services_server import *
    from service_adaptor import ServiceAdaptor21
    adapt = ServiceAdaptor21.adapt
    # Wrap generated 2.0 class to 2.1 soap_ method behaviour
    FinancialService = ServiceAdaptor21(FinancialService)
# Create a Server implementation by inheritance
# 2.1alpha implementation
class MyFinancialService(FinancialService):
    # Make WSDL available for HTTP GET
    _wsdl = "".join(open("FinancialService.wsdl").readlines())
    @adapt
    def soap_getPV(self, ps, **kw):
        # Call the generated base class method to get appropriate
        # input/output data structures
        request, response = FinancialService.soap_getPV(self, ps, **kw)
        # Comment that out if need be to see what API these objects offer:
        #print help(request)
        #print help(response)
        # Using getters/setters
        irate = request.get_element_irate()
        CFs = request.get_element_CFSequence().get_element_CF()
        # Alternative way to access the input data
        #irate = request._irate
        #CFs = request._CFSequence._CF
        # Invoke the service worker code
        PV = self.getPV(irate, CFs)
        #print "PV:", PV
        # assign return value to response object
        response = getPVResponse(PV)

        # Another way to create a serializable response object
        #class SimpleTypeWrapper(float):
        #    typecode = response.typecode
        #response = SimpleTypeWrapper(PV)

        return request, response

    def getPV(self, irate, cashFlows):
        # worker code method
        t = 0
        PV = 0.0
        for CF in cashFlows:
            PV += (CF or 0.0) * ((irate / 100.0 + 1) ** (-t))
            t += 1
        return PV


op = OptionParser(usage="%prog [options]")
op.add_option("-l", "--loglevel", help="loglevel (DEBUG, WARN)",
              metavar="LOGLEVEL")
op.add_option("-p", "--port", help="HTTP port",
              metavar="PORT", default=8080, type="int")
```

```
op.add_option("-t", "--trace", help="trace function/method calls",
              action="store_true")
op.add_option("-n", "--names", help="trace function/method names",
              metavar="NAME", action="append")
options, args = op.parse_args()


if options.trace:
    from ZSI import parse
    from inject_tracing import enhance_module
    enhance_module(ServiceContainer, names=options.names)
    enhance_module(parse, names=options.names)

# set the loglevel according to cmd line arg
if options.loglevel:
    loglevel = eval(options.loglevel, logging.__dict__)
    logger = logging.getLogger("")
    logger.setLevel(loglevel)

# Run the server with a given list of services
ServiceContainer.AsServer(port=options.port, services=[MyFinancialService(),])
```

This will print way more tracing output for incoming service requests:

```
$ /apps/pydev/hjoukl/bin/python2.4 ./myTracedFinancialServer.py -t
*** ZSI version (2, 1, 0) ***
--> AsServer(args=(), kwargs={'services': [<__main__.MyFinancialService in-
stance at 0x4c3558>], 'port': 8080})
--> server_bind(args=(<ZSI.ServiceContainer.ServiceContainer in-
stance at 0x4c35f8>,), kwargs={}) <-- server_bind()
--> setNode(args=(<ZSI.ServiceContainer.ServiceContainer in-
stance at 0x4c35f8>, <__main__.MyFinancialService in-
stance at 0x4c3558>), kwargs={})
--
> getPost(args=(<__main__.MyFinancialService instance at 0x4c3558>,), kwargs={})
<-- getPost()
<-- setNode()
--> handle(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>,), kwargs={})
--> handle_one_request(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>,), kwargs={})
--> parse_request(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>,), kwargs={})
<-- parse_request()
--> do_POST(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>,), kwargs={})
--> <lambda>(args=(<xml.dom.minidom.Document instance at 0x4c3ad0>,), kwargs={})
<-- <lambda>()
--> _check_for_legal_children(args=(<ZSI.parse.ParsedSoap in-
stance at 0x4c3940>, 'Envelope', <DOM Element: SOAP-
ENV:Envelope at 0x4c3da0>), kwargs={})
--> _check_for_legal_children(args=(<ZSI.parse.ParsedSoap in-
stance at 0x4c3940>, 'Envelope', <DOM Element: SOAP-
ENV:Envelope at 0x4c3da0>), kwargs={})
--> <lambda>(args=(<DOM Element: SOAP-ENV:Envelope at 0x4c3da0>,), kwargs={})
<-- <lambda>()
<-- _check_for_legal_children()
<-- _check_for_legal_children()
--> <lambda>(args=(<DOM Element: SOAP-ENV:Envelope at 0x4c3da0>,), kwargs={})
<-- <lambda>()
```

```
--> _valid_encoding(args=(<DOM Element: SOAP-
ENV:Envelope at 0x4c3da0>,), kwargs={})
<-- _valid_encoding()
--> <lambda>(args=(<DOM Element: SOAP-ENV:Envelope at 0x4c3da0>,), kwargs={})
<-- <lambda>()
--> _check_for_legal_children(args=(<ZSI.parse.ParsedSoap in-
stance at 0x4c3940>, 'Body', <DOM Element: SOAP-
ENV:Body at 0x4c70f8>, 0), kwargs={})
--> _check_for_legal_children(args=(<ZSI.parse.ParsedSoap in-
stance at 0x4c3940>, 'Body', <DOM Element: SOAP-
ENV:Body at 0x4c70f8>, 0), kwargs={})
--> <lambda>(args=(<DOM Element: SOAP-ENV:Body at 0x4c70f8>,), kwargs={})
<-- <lambda>()
<-- _check_for_legal_children()
<-- _check_for_legal_children()
--> <lambda>(args=(<DOM Element: SOAP-ENV:Body at 0x4c70f8>,), kwargs={})
<-- <lambda>()
--> _check_for_pi_nodes(args=(<ZSI.parse.ParsedSoap in-
stance at 0x4c3940>, [<DOM Element: ns2:getPV at 0x4c7148>], 0), kwargs={})
--> _check_for_pi_nodes(args=(<ZSI.parse.ParsedSoap in-
stance at 0x4c3940>, [<DOM Element: ns2:getPV at 0x4c7148>], 0), kwargs={})
--> <lambda>(args=(<DOM Element: ns2:getPV at 0x4c7148>,), kwargs={})
<-- <lambda>()
--> <lambda>(args=(<DOM Element: ns2:CFSequence at 0x4c7260>,), kwargs={})
<-- <lambda>()
--> <lambda>(args=(<DOM Element: ns2:CF at 0x4c7328>,), kwargs={})
<-- <lambda>()
--> <lambda>(args=(<DOM Text node "108">,), kwargs={})
<-- <lambda>()
--> <lambda>(args=(<DOM Element: ns2:CF at 0x4c72b0>,), kwargs={})
<-- <lambda>()
--> <lambda>(args=(<DOM Text node "-100">,), kwargs={})
<-- <lambda>()
--> <lambda>(args=(<DOM Element: ns2:irate at 0x4c7198>,), kwargs={})
<-- <lambda>()
--> <lambda>(args=(<DOM Text node "7">,), kwargs={})
<-- <lambda>()
<-- _check_for_pi_nodes()
<-- _check_for_pi_nodes()
--> _valid_encoding(args=(<DOM Element: SOAP-ENV:Body at 0x4c70f8>,), kwargs={})
<-- _valid_encoding()
--> <lambda>(args=(<DOM Element: SOAP-ENV:Body at 0x4c70f8>,), kwargs={})
<-- <lambda>()
--> <lambda>(args=(<DOM Element: ns2:getPV at 0x4c7148>,), kwargs={})
<-- <lambda>()
--> _valid_encoding(args=(<DOM Element: ns2:getPV at 0x4c7148>,), kwargs={})
<-- _valid_encoding()
--> <lambda>(args=(<DOM Element: SOAP-ENV:Body at 0x4c70f8>,), kwargs={})
<-- <lambda>()
--> _check_for_pi_nodes(args=(<ZSI.parse.ParsedSoap in-
stance at 0x4c3940>, [], 0), kwargs={})
--> _check_for_pi_nodes(args=(<ZSI.parse.ParsedSoap in-
stance at 0x4c3940>, [], 0), kwargs={})
<-- _check_for_pi_nodes()
<-- _check_for_pi_nodes()
--> _Dispatch(args=(<ZSI.parse.ParsedSoap in-
stance at 0x4c3940>, <ZSI.ServiceContainer.ServiceContainer in-
stance at 0x4c35f8>, <bound method SOAPRe-
questHandler.send_xml of <ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>>, <bound method SOAPRe-
questHandler.send_fault of <ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>>), kwargs={'action': ■, 'post': '/FinancialService'})
--> getNode(args=(<ZSI.ServiceContainer.ServiceContainer in-
stance at 0x4c35f8>, '/FinancialService'), kwargs={})
<-- getNode()
--> authorize(args=(<__main__.MyFinancialService in-
stance at 0x4c3558>, None, '/FinancialService', ■), kwargs={})
<-- authorize()
--> getOperation(args=(<__main__.MyFinancialService in-
stance at 0x4c3558>, <ZSI.parse.ParsedSoap instance at 0x4c3940>, ■), kwargs={})
--> getOperationName(args=(<__main__.MyFinancialService in-
stance at 0x4c3558>, <ZSI.parse.ParsedSoap instance at 0x4c3940>, ■), kwargs={})
--> <lambda>(args=(<DOM Element: ns2:getPV at 0x4c7148>,), kwargs={})
<-- <lambda>()
<-- getOperationName()
<-- getOperation()
--> Parse(args=(<ZSI.parse.ParsedSoap instance at 0x4c3940>, <FinancialSer-
vice_types.getPV_Dec object at 0x4e90b0>), kwargs={})
--> Parse(args=(<ZSI.parse.ParsedSoap instance at 0x4c3940>, <FinancialSer-
vice_types.getPV_Dec object at 0x4e90b0>), kwargs={})
<-- Parse()
<-- Parse()
--> verify(args=(<__main__.MyFinancialService in-
stance at 0x4c3558>, <ZSI.parse.ParsedSoap instance at 0x4c3940>), kwargs={})
<-- verify()
--> serialize(args=(<ZSI.writer.SoapWriter in-
stance at 0x4c77b0>, 0.93457943925233167), kwargs={})
--> writeNSdict(args=(<ZSI.writer.SoapWriter in-
stance at 0x4c77b0>, {}), kwargs={})
<-- writeNSdict()
<-- serialize()
--> sign(args=(<__main__.MyFinancialService in-
stance at 0x4c3558>, <ZSI.writer.SoapWriter instance at 0x4c77b0>), kwargs={})
<-- sign()
--> close(args=(<ZSI.writer.SoapWriter instance at 0x4c77b0>,), kwargs={})
<-- close()
--> send_xml(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>,
'<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header><SOAP-ENV:Body
xmlns:ns1="http://services.zsiserver.net/FinancialService_NS">
<ns1:PV>0.934579</ns1:PV>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>'), kwargs={})
--> send_response(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>, 200), kwargs={})
--> log_request(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>, 200), kwargs={})
--> log_message(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>, '"%s" %s %s', 'POST /FinancialService HTTP/1.1', '200', '-
'), kwargs={})
--> address_string(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>,), kwargs={})
<-- address_string()
--> log_date_time_string(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>,),
kwargs={})
```

```
<-- log_date_time_string()
adevp02 - - [11/Jan/2008 14:41:24] "POST /FinancialService HTTP/1.1" 200 -
<-- log_message()
<-- log_request()
--> version_string(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>,), kwargs={})
<-- version_string()
--> send_header(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>, 'Server', 'ZSI/1.1 BaseHTTP/0.3 Python/2.4.4'), kwargs={})
<-- send_header()
--> date_time_string(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>,), kwargs={})
<-- date_time_string()
--> send_header(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>, 'Date', 'Fri, 11 Jan 2008 13:41:24 GMT'), kwargs={})
<-- send_header()
<-- send_response()
--> send_header(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>, 'Content-type', 'text/xml; charset="utf-8"'), kwargs={})
<-- send_header()
--> send_header(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>, 'Content-Length', '456'), kwargs={})
<-- send_header()
--> end_headers(args=(<ZSI.ServiceContainer.SOAPRequestHandler in-
stance at 0x4c37b0>,), kwargs={})
<-- end_headers()
<-- send_xml()
<-- _Dispatch()
<-- do_POST()
<-- handle_one_request()
<-- handle()
```

B.2 Logging the client request

While section B.1 presents a way to enable a rather detailed tracing of method calls and arguments, it is often desirable to log the incoming client request on the server side. Unfortunately, ZSI currently offers no standard way to do so. The straightforward way to achieve this is to implement a custom request handler by copying over the `ServiceContainer.SOAPRequestHander.do_POST()` method implementation and insert debugging statements where desired. We use another approach to get at the incoming XML:

```
from optparse import OptionParser
# Import the ZSI stuff you'd need no matter what
from ZSI.wstools import logging
from ZSI import ServiceContainer
from ZSI import resolvers

from ZSI.version import Version as zsiversion
print "*** ZSI version %s ***" % (zsiversion,)
LOGGER = logging.getLogger("")
# Some more or less nifty stuff for more extensive tracing
from inject_tracing import enhance_module, TraceLocals

# Import the generated Server Object
try:
    # 2.1alpha
    from FinancialService_server import *
    # Dummy implementation
    def adapt(method):
        return method
except ImportError, e:
    # 2.0final
    from FinancialService_services_server import *
    from service_adaptor import ServiceAdaptor21
    adapt = ServiceAdaptor21.adapt
    # Wrap generated 2.0 class to 2.1 soap_ method behaviour
    FinancialService = ServiceAdaptor21(FinancialService)

# Create a Server implementation by inheritance
# 2.1alpha implementation
class MyFinancialService(FinancialService):

    # Make WSDL available for HTTP GET
    _wsdl = "".join(open("FinancialService.wsdl").readlines())

    @adapt
    def soap_getPV(self, ps, **kw):
        # Call the generated base class method to get appropriate
        # input/output data structures
        request, response = FinancialService.soap_getPV(self, ps, **kw)
        # Comment that out if need be to see what API these objects offer:
        #print help(request)
        #print help(response)
        # Using getters/setters
        irate = request.get_element_irate()
        CFs = request.get_element_CFSequence().get_element_CF()
        # Alternative way to access the input data
        #irate = request._irate
        #CFs = request._CFSequence._CF
        # Invoke the service worker code
        PV = self.getPV(irate, CFs)
        #print "PV:", PV

        # assign return value to response object
        response = getPVResponse(PV)
        # Another way to create a serializable response object
        #class SimpleTypeWrapper(float):
        #    typecode = response.typecode
        #response = SimpleTypeWrapper(PV)

        return request, response

    def getPV(self, irate, cashFlows):
        # worker code method
        t = 0
        PV = 0.0
        for CF in cashFlows:
            PV += (CF or 0.0) * ((irate / 100.0 + 1) ** (-t))
            t += 1
        return PV
# A traced request handler class that debug-logs the payload XML of the request
# message, as received from the client
class DebugSOAPRequestHandler(ServiceContainer.SOAPRequestHandler):
    @TraceLocals.trace(["xml"], logger=LOGGER)
    def do_POST(self):
        ServiceContainer.SOAPRequestHandler.do_POST(self)


op = OptionParser(usage="%prog [options]")
op.add_option("-l", "--loglevel", help="loglevel (DEBUG, WARN)",
              metavar="LOGLEVEL")
op.add_option("-p", "--port", help="HTTP port",
              metavar="PORT", default=8080, type="int")
op.add_option("-t", "--trace", help="trace function/method calls",
              action="store_true")
op.add_option("-n", "--names", help="trace function/method names",
              metavar="NAME", action="append")
options, args = op.parse_args()
if options.trace:
    from ZSI import parse
    enhance_module(ServiceContainer, names=options.names)
    enhance_module(parse, names=options.names)
# set the loglevel according to cmd line arg
if options.loglevel:
    loglevel = eval(options.loglevel, logging.__dict__)
    LOGGER.setLevel(loglevel)
# Run the server with a given list of services
address = (■, options.port)
sc = ServiceContainer.ServiceContainer(
    address, services=[MyFinancialService(),],
    RequestHandlerClass=DebugSOAPRequestHandler)
sc.serve_forever()
```

Comments:

- Instead of the `AsServer()` function, a ServiceContainer instance with a custom request handler class `DebugSOAPRequestHandler` gets used
- `DebugSOAPRequestHandler` features a decorated `do_POST()` method that calls its super-class´ `do_POST()`. The decorator provides debugging of local variables, at the point of leaving the method, and can be parameterized to print only certain variable names (we want the “xml” string only here).

This is how the extended `inject_tracing.py` module now looks like:[15]

```
import types
# Based on "Method enhancement" Python cookbook recipee by Ken Seehof
# (http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/81982)}
def enhance_method(cls, methodname, decorate):
    'replace a method with an enhancement'
    method = getattr(cls, methodname)
    _f = decorate(method)
    setattr(cls, methodname, types.MethodType(_f, None, cls))
# loop over class dict and call enhance_method() function
# for all methods to modify
def enhance_all_methods(cls, decorate):
    for methodname in cls.__dict__:
        if not methodname.startswith("__") \
           or methodname in ["__call__", "__call___"]:
            method = getattr(cls, methodname)
            if isinstance(method, types.MethodType):
                enhance_method(cls, methodname, decorate)
def traced(enter_text="-->", leave_text="<--"):
    """Configurable Function decorator.
    Outputs the text given as enter_text plus the function name and arguments
    before entering the decorated function, and the leave_text string after
    having left the decorated function.
    Example:
    >>> @traced("into", "out of")
    ... def foo(x,y):
    ...     print x,y
    ...     return x
```

[15]Based on Mailing list posting http://mail.python.org/pipermail/python-list/2005-April/319662.html, by Bengt Richter

```
    ...
    >>> foo(1, 2)
    into foo(args=(1, 2), kwargs={})
    1 2
    out of foo()
    1
    >>>
    """
    def traced(f):
        _f = f
        def f(*args, **kwargs):
            print enter_text, "%s(args=%s, kwargs=%s)" % (_f.__name__, args,
                                                          kwargs)
            try:
                return _f(*args, **kwargs)
            finally:
                print leave_text, "%s()" % (_f.__name__)
        f.__name__ = _f.__name__
        return f
    return traced
def enhance_module(module, decorator=traced, names=[]):
    """Wrap module class methods and callables with given decorator. Applies
    to list of names only, if given, otherwise to all detected
    methods/callables.
    """
    for objname, obj in module.__dict__.items():
        if not names or objname in names:
            if isinstance(obj, (type, types.ClassType)):
                #print obj
                enhance_all_methods(obj, decorator())
            elif callable(obj):
                #print obj
                module.__dict__[objname] = decorator()(obj)
# Based on Python mailing list post
# http://mail.python.org/pipermail/python-list/2005-April/319662.html
# as posted by Bengt Richter
class TraceLocals(object):
    """Tracing of local function or method variable names, as of leaving the
    function.
    Can be used standalone or easiest by applying the method decorator static
    method "trace".
    """
    from sys import settrace

    def __init__(self, fnames=[], vnames=[], logger=None):
        """TraceLocals constructor.
        Parameters:
            fnames: List of function or method names to be traced.
            vnames: List of local variable names to output. If list is
                empty, all local variables will be printed.
        """
        self.fnames = set(fnames)
        self.vnames = set(vnames)
        if logger:
            self._print = logger.debug

    def _gwatch(self, frame, event, arg):
        """
        Global scope watcher. When a new scope is entered, returns the local
        scope watching method _lwatch to do the work for that.
        """
```

```
            if event == 'call':
                name = frame.f_code.co_name   # name of executing scope
                if name in self.fnames:
                    return self._lwatch

        def _lwatch(self, frame, event, arg):
            if event == 'return':
                vnames = self.vnames or frame.f_locals.keys()
                for vname in vnames:
                    if frame.f_locals.has_key(vname):
                        self._print("%s: %s = %s",
                                    frame.f_code.co_name, vname,
                                    frame.f_locals[vname])
            else:
                return self._lwatch # keep watching for return event

        def on(self):
            """Set the system trace hook to enable tracing.
            """
            self.settrace(self._gwatch)

        def off(self):
            """Reset the system trace hook to disable tracing.
            """
            self.settrace(None)

        def _print(self, msg, *args):
            print msg % args

        @staticmethod
        def trace(vnames=[], logger=None):
            """Method/function decorator that enables tracing of local variables, as
            of leaving the function.
            Parameters:
                vnames: List of variable names that shall be traced. If list is
                    empty, all local variables will be printed.
                logger: Optional logger instance. If given, ouput will be routed
                    through logger's debug method.
            """
            def _decorate(func):
                tracer = TraceLocals([func.__name__, ], vnames, logger=logger)
                def _f(*args, **kwargs):
                    tracer.on()
                    try:
                        return func(*args, **kwargs)
                    finally:
                        tracer.off()
                _f.__name__ = func.__name__
                return _f
            return _decorate
```

Using this, the ZSI server now logs the incoming client message:

```
$ /apps/pydev/hjoukl/bin/python2.4 myTracedFinancialServer.py -l "DEBUG"
*** ZSI version (2, 1, 0) ***
----  ZSI.TCcompound.ComplexType  ----
    [DEBUG]   parse
    [DEBUG]   ofwhat: (<ZSI.TCnumbers.FPfloat instance at 0x4bd210>, <Finan-
cialService_types.CFSequence_Dec object at 0x486df0>)
    [DEBUG]   what: (http://services.zsiserver.net/FinancialService_NS,irate)
    [DE-
BUG]   child node: (http://services.zsiserver.net/FinancialService_NS,ns1:irate)
    [DE-
BUG]   what: (http://services.zsiserver.net/FinancialService_NS,CFSequence)
    [DE-
BUG]   child node: (http://services.zsiserver.net/FinancialService_NS,ns1:CFSequence)
    [DEBUG]   parse
    [DEBUG]   ofwhat: (<ZSI.TCnumbers.FPfloat instance at 0x4bd260>,)
    [DEBUG]   what: (http://services.zsiserver.net/FinancialService_NS,CF)
    [DE-
BUG]   child node: (http://services.zsiserver.net/FinancialService_NS,ns1:CF)
    [DE-
BUG]   child node: (http://services.zsiserver.net/FinancialService_NS,ns1:CF)
adevp02 - - [18/Jan/2008 13:38:41] "POST /FinancialService# HTTP/1.1" 200 -
----    ----
    [DEBUG]   do_POST: xml = <SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:ns1="http://services.zsiserver.net/FinancialService_NS">
<ns1:getPV>
<ns1:irate>7.000000</ns1:irate>
<ns1:CFSequence><ns1:CF>-100.000000</ns1:CF>
<ns1:CF>110.000000</ns1:CF>
</ns1:CFSequence></ns1:getPV>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

References

[1] Salz, Rich; Blunck, Christopher: ZSI: The Zolera Soap Infrastructure. <http://downloads.sourceforge.net/pywebsvcs/zsi-2.0.pdf>, Release 2.0.0, February 01, 2007.

[2] Boverhof, Joshua; Moad, Charles: ZSI: The Zolera Soap Infrastructure User´s Guide. <http://downloads.sourceforge.net/pywebsvcs/guide-zsi-2.0.pdf>, Release 2.0.0, February 01, 2007.

[3] Salz, Rich; Blunck, Christopher: ZSI: The Zolera Soap Infrastructure. <http://downloads.sourceforge.net/pywebsvcs/ZSI-2.1-a1.tar.gz>, Release 2.1.0, November 01, 2007.

[4] Boverhof, Joshua; Moad, Charles: ZSI: The Zolera Soap Infrastructure User´s Guide. <http://downloads.sourceforge.net/pywebsvcs/ZSI-2.1-a1.tar.gz>, Release 2.1.0, November 01, 2007.

[5] van Engelen, Robert: gSOAP 2.7.3 User Guide. <http://www.cs.fsu.edu/˜engelen/soap.html>, June 27, 2005.

[6] Butek, Russell: Which style of WSDL should I use? <http://www-106.ibm.com/developerworks/webservices/library/ws-whichwsdl>, May 24, 2005.

[7] Joukl, Holger: Interoperable WSDL/SOAP web services introduction: Python ZSI, Excel XP, gSOAP C/C++ & Applix SS). <http://pywebsvcs.sourceforge.net/holger.pdf>. July 22, 2005.

[8] Hobbs, Chris: Using ZSI. <http://pywebsvcs.sourceforge.net/cookbook.pdf>, August 1, 2007.

[9] Locad.com: Web Services Tutorial with Python. <http://www.lokad.com/web-services-time-series-forecasting-tutorial-python.ashx>, January 2, 2008.